1991-09

# An intelligent training system for helicopter recognition

Ling, Ming-Tien

Monterey, California. Naval Postgraduate School

# NAVAL POSTGRADUATE SCHOOL
## Monterey, California

# THESIS

**AN INTELLIGENT TRAINING SYSTEM
FOR
HELICOPTER RECOGNITIOIN**

by

Ming-Tien Ling

September 1991

Thesis Advisor: Dr. Yuh- jeng Lee

Approved for public release; distribution is unlimited.

# REPORT DOCUMENTATION PAGE

| | |
|---|---|
| 1a. REPORT SECURITY CLASSIFICATION: UNCLASSIFIED | 1b. RESTRICTIVE MARKINGS |
| 2a. SECURITY CLASSIFICATION AUTHORITY | 3. DISTRIBUTION/AVAILABILITY OF REPORT: Approved for public release; distribution is unlimited |
| 2b. DECLASSIFICATION/DOWNGRADING SCHEDULE | |
| 4. PERFORMING ORGANIZATION REPORT NUMBER(S) | 5. MONITORING ORGANIZATION REPORT NUMBER(S) |
| 6a. NAME OF PERFORMING ORGANIZATION: Computer Science Dept. Naval Postgraduate School; 6b. OFFICE SYMBOL (if applicable): CS | 7a. NAME OF MONITORING ORGANIZATION: Naval Postgraduate School |
| 6c. ADDRESS (City, State, and ZIP Code): Monterey, CA 93943-5000 | 7b. ADDRESS (City, State, and ZIP Code): Monterey, CA 93943-5000 |
| 8a. NAME OF FUNDING/SPONSORING ORGANIZATION; 8b. OFFICE SYMBOL (if applicable) | 9. PROCUREMENT INSTRUMENT IDENTIFICATION NUMBER |
| 8c. ADDRESS (City, State, and ZIP Code) | 10. SOURCE OF FUNDING NUMBERS: PROGRAM ELEMENT NO. / PROJECT NO. / TASK NO. / WORK UNIT ACCESSION N |

11. TITLE (Include Security Classification)
AN INTELLIGENT TRAINING SYSTEM FOR HELICOPTER RECOGNITION

12. PERSONAL AUTHOR(S)
Ming-Tien Ling

| 13a. TYPE OF REPORT | 13b. TIME COVERED | 14. DATE OF REPORT (Year, Month, Day) | 15. PAGE COUNT |
|---|---|---|---|
| Master's Thesis | FROM TO | September 1991 | 166 |

16. SUPPLEMENTARY NOTATION The views expressed in this thesis are those of the author and do not reflect the official policy or position of the Department of Defense or the United States Government.

| 17. COSATI CODES | | | 18. SUBJECT TERMS (Continue on reverse if necessary and identify by block number) |
|---|---|---|---|
| FIELD | GROUP | SUB-GROUP | Intelligent Computer-Assisted Instruction, Artificial Intelligence, Helicopt Recognition Training |
| | | | |
| | | | |

19. ABSTRACT (Continue on reverse if necessary and identify by block number)

This thesis investigates the design and implementation of an intelligent computer-assisted instruction system f visual helicopter training. We developed the Helicopter Recognition Tutor accordig to the four-component model a generalized intelligent computer-assisted instruction system. The tutor system provides an interactive tutoring e vironment that teaches, reviews, and tests visual helicopter recognition skills at a level relevant to the student. It i structs the student at three different levels: the novice level, the intermediate level, and the expert level, based on t student's understanding of the Wing, Engine, Tail, Fuselage, Undercarriage, and Rotor (WETFUR) features of a pa ticular helicopter.

| 20. DISTRIBUTION/AVAILABILITY OF ABSTRACT | 21. ABSTRACT SECURITY CLASSIFICATION | |
|---|---|---|
| ☒ UNCLASSIFIED/UNLIMITED ☐ SAME AS RPT. ☐ DTIC USERS | UNCLASSIFIED | |
| 22a. NAME OF RESPONSIBLE INDIVIDUAL: Yuh-jeng Lee | 22b. TELEPHONE (Include Area Code): (408) 646-2361 | 22c. OFFICE SYMBOL: CS/Le |

Approved for public release; distribution is unlimited.

An Intelligent Training System
for
Helicopter Recognition

by

Ming-Tien Ling
Captain, Republic of China Army
B.S., Chung-Cheng Institute of Technology

Submitted in partial fulfillment
of the requirements for the degree of

MASTER OF SCIENCE IN COMPUTER SCIENCE

from the

## ABSTRACT

This thesis investigates the design and implementation of an intelligent computer-assisted instruction system for visual helicopter recognition training. We developed the Helicopter Recognition Tutor according to the four-component model of a generalized intelligent computer-assisted instruction system. The tutor system provides an interactive tutoring environment that teaches, reviews, and tests visual helicopter recognition skills at a level relevant to the student. It instructs the student at three different levels: the novice level, the intermediate level, and the expert level, based on the student's understanding of the Wing, Engine, Tail, Fuselage, Undercarriage, and Rotor (WETFUR) features of a particular helicopter.

## TABLE OF CONTENTS

# I. INTRODUCTION

## A. BACKGROUND

Computer-based instruction systems have been under development since the early 1960s. The primary characteristic of these systems is to provide an interactive learning environment for all students in all levels of education and training. The needs of each student can be satisfied by individualizing the learning experience. In most of today's classrooms, we have relatively few opportunities for students to receive direct attention from the instructors. Although human tutors can provide the one-to-one tutoring, they may be too costly. Computer-based instruction, however, can integrate the knowledge of many experts and can provide interactive, individualized instruction at reasonable cost.

Although computer-based instruction has such massive potential, it has not been utilized as extensively as it could have been. One of reasons for this is the absence of ability to inference or reason. To successfully teach, computers must be able to capture the skill of inferencing or reasoning.

Therefore, several computer-assisted instruction (CAI) researchers, who have a background in artificial intelligence (AI), applied various AI techniques to the CAI system. The intention of their effort is to develop a new class of instruction system, the intelligent computer-assisted instruction (ICAI) system.

In developing an ICAI system, the most important part to consider is choosing an appropriate way of representing the knowledge to be taught. Another major consideration

is how to model the student's current understanding of subject matters. The ICAI system then reasons from one item of knowledge to another by using inferencing procedures. This thesis will investigate the possible use of an ICAI system for training soldiers in helicopter recognition.

## B. OBJECTIVE

Helicopters have become more important in military combat since the Vietnam War. Nowadays, helicopters are widely used in various military missions, for example, transporting personnel and cargo, attacking the enemy, and so on. Therefore, the ability for the soldier to recognize helicopters becomes one of most important training missions in the military. Although there exist several training programs for visual aircraft or helicopter recognition today, these programs usually train the soldier by using photos, drawings, and slides. Moreover, visual helicopter recognition skills are not easy to teach or to learn. In order to fill a gap that exists in most visual helicopter recognition training programs, this thesis is an attempt to develop a useful intelligent computer-based training tool using current AI techniques. The Helicopter Recognition Tutor, which we developed over the past months, is the product of this thesis. The tutor is designed for training purposes. It can identify a soldier's current ability to recognize existing helicopters in operation, and then teach at a level appropriate to that ability. Therefore, it is useful for introducing visual helicopter recognition to new soldiers and for providing refresher training to more advanced soldiers.

## C. ORGANIZATION

Chapter II provides an overview of intelligent computer-assisted instruction systems. We review the model of a generalized ICAI system and some applications in the ICAI field. Chapter III provides an overview of helicopter recognition and current training methods. Chapter IV describes the design and implementation of the Helicopter Recognition Tutor. The results and evaluation of the tutor are also discussed in Chapter IV. Finally, Chapter V is the conclusion and discusses our achievements and recommendations for future work. Appendix A contains a user's manual for the tutor. Appendix B contains the source code for the tutor. Appendix C provides some helicopter images files which are used in the tutor.

# II. OVERVIEW OF INTELLIGENT COMPUTER-ASSISTED INSTRUCTION SYSTEM

## A. INTRODUCTION

Since the early 1960s, computer technology has been applied in all levels of education. Many educational applications of computer technology have been developed. These applications include grading tests and solving the course scheduling problem. In addition, computers are also uses for educational purposes outside schools. For example, the military uses computers for training. The industry uses computers to design and manufacture products. Computer application in education was generally labeled computer-assisted instruction (CAI). Intelligent computer-assisted instruction (ICAI) systems have been developed because there were many flaws in CAI systems. These flaws have been discussed in the ICAI literature [Refs.1, 2]. Limitations were enumerated in detail by Marlene Jones [Ref. 3]. In her paper, "Applications of Artificial Intelligence within Education", she asserts that most CAI systems have the following limitations:

- unable to understand the subject matter being taught.
- unable to decide (reason about) what should be taught next.
- unable to foresee, diagnose, and understand the student's misconceptions.
- unable to improve or update current teaching strategies or learn new one.
- unable to do conversations with students in the student's natural language.

Some researchers working in the fields of machine learning, knowledge representation, natural language understanding, and expert system, realized the flaws of CAI systems and sought to incorporate Artificial Intelligence (AI) to improve the quality and effectiveness of CAI. The consequence of their combined effort, is a new class of instruction system, the ICAI system.

## B. INTELLIGENT COMPUTER-ASSISTED INSTRUCTION SYSTEMS

In this section, we will introduce the components of an ICAI system and some applications that have been developed in ICAI systems.

### 1. Components

The typical model of an ICAI system includes the following components: the expert model, the student model, the tutor model, and the communication model [Ref. 4]. Figure 2.1 depicts a generalization of the components of an ICAI system.

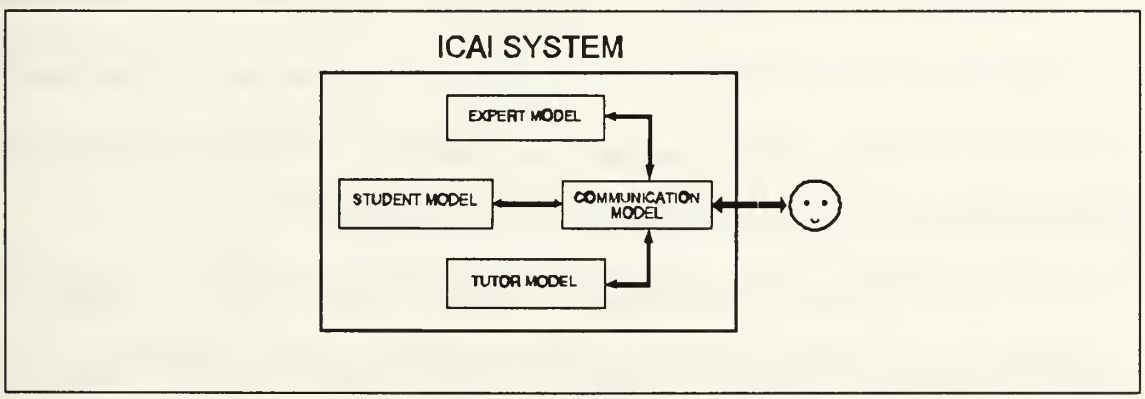



**Figure 2.1 Components of an ICAI System**

### *a. Expert Model*

The expert model is the component of the system where the expertise and information concerning the subject matter is represented within the ICAI system. It serves two distinct functions. First, the expert model is the source of the knowledge to be imparted and tested. Second, the expert model provides a means of evaluating the student's responses [Ref. 5].

In order to develop an effective ICAI system, the domain knowledge of the expert model should be the knowledge of several experts of that field. In addition, it is important to select an appropriate representation technique.

### *b. Student Model*

The student model is the component of the system in which information concerning what the student knows and does not know, and the student's understanding of material being taught is gathered and represented. The purpose of the student model is to judge the student's knowledge. In addition, it forms a hypothesis for the evaluation of the student's misconceptions and reasoning strategies. Thereby the tutor model can make appropriate suggestions to correct any misconceptions.

In many ICAI systems, the student model is built by comparing the student's responses to that of the expert. In other words, the model represents the student's knowledge as a subset of the expert's knowledge. This model is referred to as an "overlay model". Another model called the "buggy" model represents the student's misconceptions about the knowledge domain. [Ref. 6]

### *c. Tutor Model*

The tutor model contains the theory of the teaching method used by the system. This model requires the subject knowledge and the teaching strategies. There are many teaching strategies, however two types are primarily used in the ICAI systems. These two types are the Socratic method and coaching method. In the Socratic method, the student is given questions which guide them through the process of debugging their misconceptions. On the other hand, the coaching method encourages learning through activities such as computer games. The immediate aim of the student is to have fun, learning and skill acquisition is an indirect result. The computer "coach" observes the progress of the game and offers new information or suggestions as needed. Tutoring occurs through appropriate interruptions by the "coach". A successful computer "coach" should be able to determine what skills or knowledge the student is likely to acquire based on their playing styles. It should also be able to judge effective means to intercede during playing game and offer advise [Ref. 7].

### *d. Communication Model*

The communication model represents the medium which interfaces with the student. This model serves two important functions. First, the model interprets the student's responses. Second, the model displays the system's output on the screen [Ref. Park 1987]. Most of the early ICAI systems did not contain an explicit communication model. However, in recent years, the development of sophisticated and expressive graphical interface and improved natural language understanding system have made the communication model an indispensable part of any ICAI system.

### 2. Application Examples

**SCHOLAR** was one of the earliest ICAI systems. It was developed in 1970. The **SCHOLAR** program is a mixed-initiative computer-based tutor that teaches students about South American geography. Both the student and the tutor system could start a conversation by asking questions. It was a pioneering effort in the development of programs which handle unanticipated questions and generate instructional material in varying levels of detail, based on the context of the dialogue [Ref. 7].

**WHY** tutors students in the causes of rainfall. **WHY** is an extension of **SCHOLAR**. It also implements the Socratic method. It was developed to fill a requirement for systems which handle subject matters that are not factual in nature. The student's errors could involve not only forgotten facts, but also misconceptions about why processes work the way they do. [Ref. 7]

**SOPHIE** was a tutor system that teaches electronic circuit diagnosis. **SOPHIE** allows students to learn and acquire problem solving skills by trying out their ideas, rather than by instructing. The system contains a model of the problem solving knowledge in the domain and heuristic strategies for answering the student's questions and dealing with misconceptions. The students are challenged to explore their own ideas and develop conjectures or hypothesis about troubleshooting strategies for electronic circuits. The function of the expert is to provide detailed feedback concerning the logical validity of the student's proposed solutions. [Refs. 5, 7]

**WEST** was the first ICAI system. It was developed to tutor students in elementary arithmetic. The coaching method used in the **WEST** system provides a

computer-based learning environment where the student is involved in an activity, like playing a computer game. The "coach" observes in the background during the game and sometime offers suggestions for improvement [Ref. 7]. In addition, **WUMPUS** was another system which use the coaching strategy. The system challenges the student in domain of logic, probability, decision theory and geometry [Ref. 7].

**GUIDON** is an adaptation of the **MYCIN** expert system for medical diagnosis. Its mix-initiative dialogue differs from that of other ICAI systems in its use of prolonged and structured interactions. It does more than simply respond to the student's last action (as in **WEST** and **WUMPUS**) and repetitive questioning and answering (as in **SCHOLAR** and **WHY**). **GUIDON** showed that we could treat the tutoring knowledge as a rule-based system itself by moving tutoring knowledge apart from subject knowledge. [Ref. 7]

**BUGGY** was developed to determine correctly the student's misconceptions about basic mathematical skills. The idea of the **BUGGY** tutor is attempting to identify the misconceptions the student may have in solving a problem by using a catalog of common problems. The system categorizes common mathematical errors. When the student makes a mistake, the system tries to identify errors by matching the procedure which the student used in solving the problem with its knowledge domain of "buggy" procedures. Once the match is made, the tutor can explain the student's errors and teach in such manner that help the student to understand the problem as well as its solutions.

The **Aircraft Recognition Tutor** was developed for the aircraft recognition training in the military. The tutor system was designed and implemented according to

the four component models of a generalized ICAI system. It uses the coaching method and provides a "learn by fun" training environment, such as a computer game. Its domain knowledge and the tutorial knowledge are well structured and good for the fixed-wing aircraft recognition training. The system also provides the simple and intuitive user's interface for the user to communicate with the system. The user interface is not attractive, since the tutor uses the CGA 640x200 two color modes. Because helicopter recognition is taught in the similar manner as fixed-wing aircraft recognition and the system model is well structured and modular, the tutor system can be easily modified to be suitable for the helicopter recognition training. In this thesis, these modifications have been done by defining the features that helicopters possess and defining the helicopter objects. In other words, we creates a new knowledge domain for helicopters. In addition, we uses the VGA 640x480 sixteen color mode to make the user interface more attractive. [Ref. 10]

## C. SUMMARY

ICAI systems are developed by applying AI techniques to CAI systems. It is seen as very possible that the role of AI in computer-based applications establishes a new type of learning environment.

The four component model of a generalized ICAI is widely accepted by current ICAI researchers. These four components are the expert model, the student model, the tutor model and the communication model. Although there are different opinions about

the particular functions that each component model should perform, there is little dispute about the entire framework of the system model.

Some applications of ICAI systems have been described in this chapter. These systems do not demonstrate that they can be used completely and effectively in training or classroom situations within the four component models. But each system does demonstrate varying amounts of completeness for each model. For example, **SOPHIE** has a good tutor model, but a poor student model [Ref. 8]. There are still many other applications which can be found in books concerning ICAI system research, such as *Artificial Intelligence and Tutoring Systems* and *The Handbook of Artificial Intelligence* [Refs 5, 7].

# III. OVERVIEW OF HELICOPTER RECOGNITION

## A. PURPOSE

Helicopters first saw service in the Korean War during the 1950s and were widely used in the Vietnam Conflict. Today, they are widely exploited in the areas of agriculture, public service, and military operations, among which the latter one is the most commonly seen. Some characteristics of helicopters include:

- little restrictions from geography or airfields,
- high mobility,
- high degree of safety.

Therefore, for military purposes, they are generally used in

- transporting cargos and personnel,
- launching surprise attacks and assaults,
- detecting submarines,
- sweeping and deploying mines.

In the foreseeable future, helicopters will play an even more important role in military combats. [Ref. 9]

Therefore, it is imperative to be able to distinguish the helicopters of different parties in a confrontation. If we cannot identify positively whether the helicopter which reaches attack range is friendly or not, it may cause heavy casualties on our side.

Therefore, it is crucial to train military personnels to recognize friendly or unfriendly helicopters in order to reduce the possible threat.

## B. BASICS OF HELICOPTER RECOGNITION

The current best method for teaching visual aircraft recognition is based on the following features of aircraft: wings to provide lift, an engine to provide motive power, a fuselage to carry the payload and controls, and a tail assembly which usually serves the purpose of controlling the direction of flight. This method is called the WEFT theory. [Ref. 10]

This theory can be applied to helicopters, except that wings appear only occasionally in stub form, the tail structure consists of rear rotors and some miniature tails, and the main rotor(s) dominates the shape [Ref. 10]. In addition, the type of helicopter undercarriage is also a key feature for recognition. In order to accomplish helicopter recognition, this thesis will slightly modify the WEFT theory to a new theory. The new theory consists of major features for helicopter recognition . It is called the WETFUR (Wing, Engine, Tail, Fuselage, Undercarriage, Rotor) theory.

The WETFUR theory of helicopters is described as follows:

- Wing: Only a few helicopters have wings, usually in stub form; most have no wings.
- Engine: Engine type and numbers play a role in identifying a particular helicopter. Engine types include turboshaft, turbofan, and piston.
- Tail: The position and the number of horizontal stabilizers can be used for helicopter recognition. The horizontal stabilizers may be located in the middle or on the end of the tail boom. The helicopter may have half, full or no horizontal stabilizers.

- Fuselage: The helicopter tail boom is the main part of fuselage. It generally can be classified as one of two kinds, the open tail boom, or the fair tail boom.

- Undercarriage: The helicopter undercarriage can be classified as the skid undercarriage, the wheeled undercarriage, and the retractable undercarriage.

- Rotor: There are two kinds of helicopter main rotors, the single rotor, and the twin rotor.

## C. TRAINING PROGRAMS

There are several aids for training personnel in visual helicopter recognition. These aids include photos, card sets, slides , movies, models and drawings. Most of these aids are used in helicopter recognition. The training programs that are often used include session, classroom training, and supplemental resources.

Training sessions in helicopter recognition can be provided to allow soldiers to become familiar with various existing helicopters. During these sessions, the instructor points out key identifying features by using slides of helicopter photos along with silhouettes or models and suggests memory aids to help the soldiers remember the helicopters along with features.

Some training units provide classroom training in helicopter recognition. In this case, a knowledgeable instructor not only introduces the features of visual helicopter recognition by using helicopter photos or silhouettes, but also supplies soldiers with comparison feedback for wrong answers. This type of training in helicopter recognition has its limitations, since the time needed for individual instruction and the number of knowledgeable instructors is usually limited.

Supplemental resources available for recognizing helicopters include books such as *Jane's World Aircraft Recognition Handbook* [Ref. 11] and flash cards, with line drawings or silhouettes.

# IV. HELICOPTER RECOGNITION TUTOR

The Helicopter Recognition Tutor was designed and implemented on an IBM compatible personal computer using the Turbo Pascal V6.0 programming language. In order for future improvements or changes to be made to the individual components of the tutor, the Object Oriented Programming (OOP) methodology was used to develop the tutor. The OOP concept gives the tutor development more structure and modularity, better abstraction, and reusability. In other words, all these features add up to code that is more structured, extensible, and easy to maintain [Ref. 12].

## A. ARCHITECTURE

### 1. Expert Model

The expert model is the component of the system in which all of the expertise and information about the subject matter is represented in an ICAI system. In the helicopter recognition tutor, the expert model is composed of the helicopter images stored in binary format and WETFUR descriptions stored in text format. Each helicopter exists in the program as a composite object, thus encompassing both the textual and binary information about the helicopter, as well as the functions and procedures that are used to operate on that information.

The helicopter images that are contained in the system came from *Jane's World Aircraft Recognition Handbook* [Ref. 11]. They were scanned in by using a

scanner with the PCX file format. These images were then brought into a software package called VGA Paint, where the resolution of image was changed to match that of the monitor in the tutoring system (VGA 640x480, 16 colors), resized, and cleaned up. Since the images still exist in the PCX format, they could be easily modified by using any PCX compatible paint program, for example, PC Paintbrush.

### 2. Student Model

The student model reflects the student's proficiency in helicopter recognition. It receives performance evaluations with respect to the knowledge that the student is expected to acquire from the tutor model. In other words, it tracks student performance information, as well as the student's current level or mode.

Each student that uses the tutor system will have his own student model. When a new student enters the system, the system first diagnoses the student's ability in helicopter recognition in order to decide the appropriate level and mode at which the student should begin. The system then assigns the student a default model which corresponds to the level, and begins to teach a session at the level. The tutor system contains a utility that allows the student model database to be accessed. This utility provides the performance information about the student.

### 3. Tutor Model

The tutor model is the component of the system which chooses problems to be solved, evaluates performance, and provides assistance. It performs two distinct function: (1) determination of the student's instructional needs and (2) selection and

presentation of domain knowledge [Ref. 13]. The tutor model in the Helicopter Recognition Tutor is composed of three user levels: Novice, Intermediate, and Expert, and three modes: Teach, Review, and Test. Figure 4.1 shows the component structure of the tutor model.

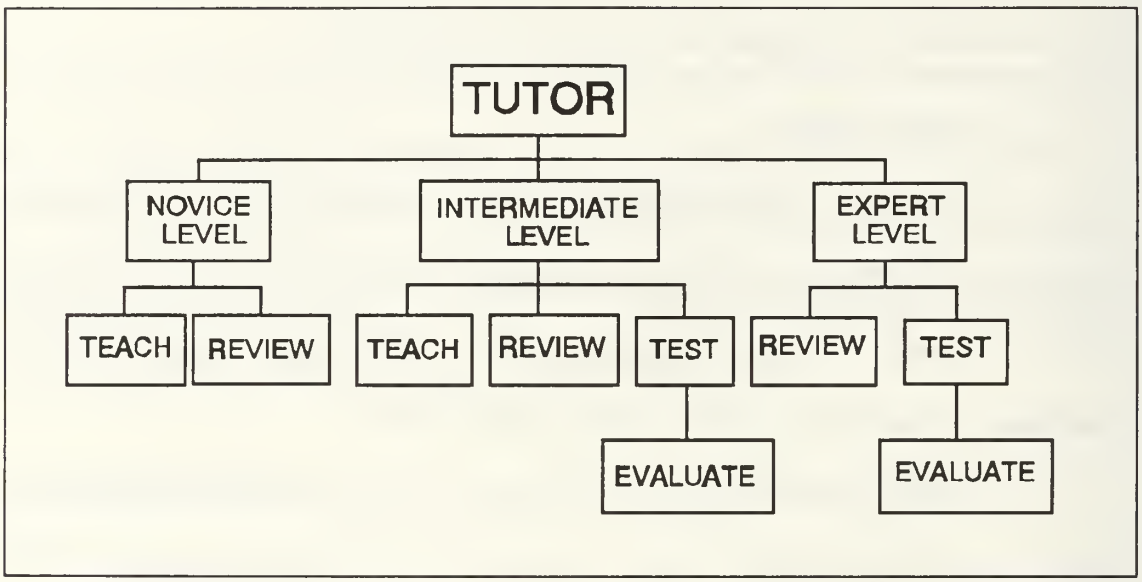



**Figure 4.1 Component Structure of the Tutor Model**

*a. Novice Level*

In this level, the tutor introduces the WETFUR features to the student by displaying a generic helicopter. In Review mode, all of the features introduced in the Teach mode are displayed randomly. The student is then expected to identify these features, and the tutor takes an appropriate course of action which is based on the student's response.

### *b. Intermediate Level*

In the intermediate level, the tutor teaches the WETFUR features of each individual helicopter by presenting a visual image of the helicopter and identifying the WETFUR features which differentiate that particular helicopter from the others. In Review mode, each helicopter is presented randomly but completely and the student is allowed to identify the helicopter by name and nomenclature. The tutor then takes action based on the student's response. In Test mode, the tutor presents the student with each helicopter from a default set of test cases, allowing the student to identify the helicopter. A record of the student's performance will be maintained by the system.

### *c. Expert Level*

In this level, no visual image of the helicopter is presented to the student. The tutor reviews and tests the student based on the WETFUR features of the helicopter. The tutor takes action based on the student's response and corrects the student's misconception. A record of the student's performance will be maintained by the system.

## 4. Communication Model

The communication model provides the tutor system with the user interface and a control program that serves to control the actions of the other three models, including interaction between these models. The interface for the system is graphically oriented. It is composed of several distinct functions: menus to receive information and choices from the student, dialogue boxes to communicate with the student, and help screens to give the student context sensitive help information. "Hot" keys allow the

students to easily quit what they are doing, request help, or interrupt the tutor. In order to encourage the students to learn, the tutor system also includes a computer game mode. Performance in the game mode of the system is not tracked in the student model.

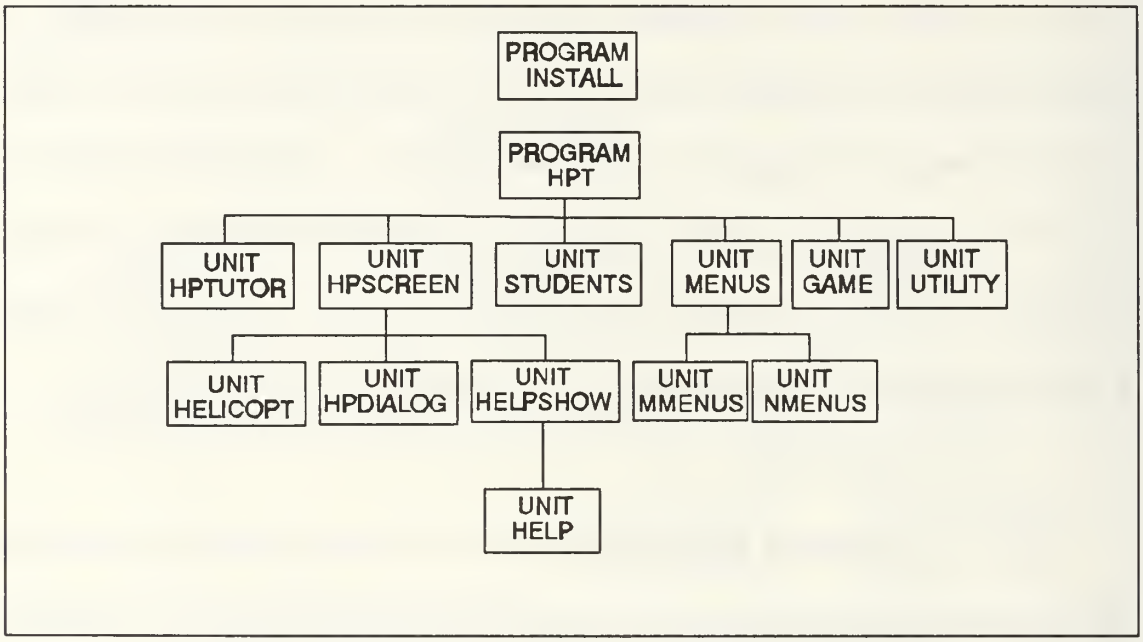



**Figure 4.2 Program Structure of the Tutor System**

## B. DETAILS OF IMPLEMENTATION

The tutor system is composed of an install program, and the tutor program. The tutor program consists of a main program and several units. Figure 4.2 shows the program structure of the Helicopter Recognition Tutor.

### 1. Program Install

This program is used for installing the Helicopter Recognition Tutor on the user's personal computer. The program provides instructions on the screen to allow the

user to complete the installation. It also check to see that the user's system meets all of the requirements for running the tutor. For instance, the program checks if a VGA adapter is present, checks for or creates a subdirectory called HP, and checks if disk space is available for the tutor.

### 2. The Tutor Program

The tutor system consists of a main program called HPT and several units used by the main program.

#### *a. Program HPT*

This program is the main program for the tutor system. It first makes sure that the user's computer is compatible with the tutor system by using the PCX utility. It also presents the main menu to the user and passes control to the other units based on the user's input.

#### *b. Unit Hpscreen*

The screen object class is defined in this unit. The purpose of this unit is to allow other classes to inherit methods and variables that they have in common.

#### *c. Unit Hpdialog*

The purpose of this unit is to define the dialogue object class. A dialogue object is an interactions dialogue box that is shown on the screen. An interaction dialogue box provides either textual messages to the user or a location for the user to input message.

*d. Unit Helicopt*

The helicopter object class is defined in this unit. Helicopter objects are composed of a record that contains the WETFUR information about the helicopter, and redefines the procedures and methods inherited from the screen object class.

*e. Unit Menus, Unit NMenus, and Unit MMenus*

The purpose of these units is to provide several menus, including a main menu, a help menu, a setup menu, a menu for helicopter, and a menu for WETFUR features for the user to select the desired item.

*f. Unit Helpshow and Unit Help*

The purpose of these two units is to provide the context sensitive help for the user. All of files containing help information were created by a paint program, VGA Paint. The format of the files is PCX file format.

*g. Unit Students*

The student model object is defined in this unit. The purpose of this unit is to create an individual student model database that contains all of the information known about each student. The information includes a student's name, current level and mode, the latest test score, and the number of helicopters shown and missed during the current level and mode. This unit also defines procedures to get, update, and save the student model, and functions to get and add entries to the model.

*h. Unit Game*

In order to encourage the user to use the tutor system, this unit is designed to provide a game mode for the user. This unit contains procedures that control the one or two player game mode.

*i. Unit Hptutor*

The purpose of this unit is to manage the teaching strategy for the helicopter recognition. This unit is composed of several procedures and functions. These include procedures to diagnose the student's level, take an action corresponding to the level where the student is, and evaluate the student's performance. This unit also defines functions that display WETFUR features of helicopters and compare WETFUR features of two helicopters.

*j. Unit Utility*

This unit defines a set of procedures. The purposes of these procedures are to allow the system administrator to select the helicopter that will be taught by the system from among those defined, to add to or modify the helicopters that are defined in the system, and to retrieve a student report.

## C. RESULTS AND EVALUATION

The Helicopter Recognition Tutor was designed to run on existing personal computers with a VGA graphics adapter. The tutor program consists of a main program and a set of units used by the main program. The source code for the tutor system is in Appendix B. The tutor system was tested on a database that included 27 helicopters

and their WETFUR features. Test runs were finished by the students that have different abilities in helicopter recognition. In Appendix A, the user's manual explains the tutor system use. The system requirements are also listed in the user's manual.

We believe that the best way to progress in helicopter recognition is to "look and learn" as frequently as possible. Since the tutor is developed for visual helicopter recognition training and can be operated on a personal computer, students may use the tutor to learn helicopter recognition as often as necessary. In addition, the game mode of the tutor provides a "learn by fun" environment and encourages frequent use of the system.

## D. A SAMPLE SESSION

The following screen displays depict a sample tutoring session in the Helicopter Recognition Tutor. Figure 4.3 shows a screen image with a main menu that provides five selections for the user. For example, the user selects the "TUTOR" item to enter the tutor mode.

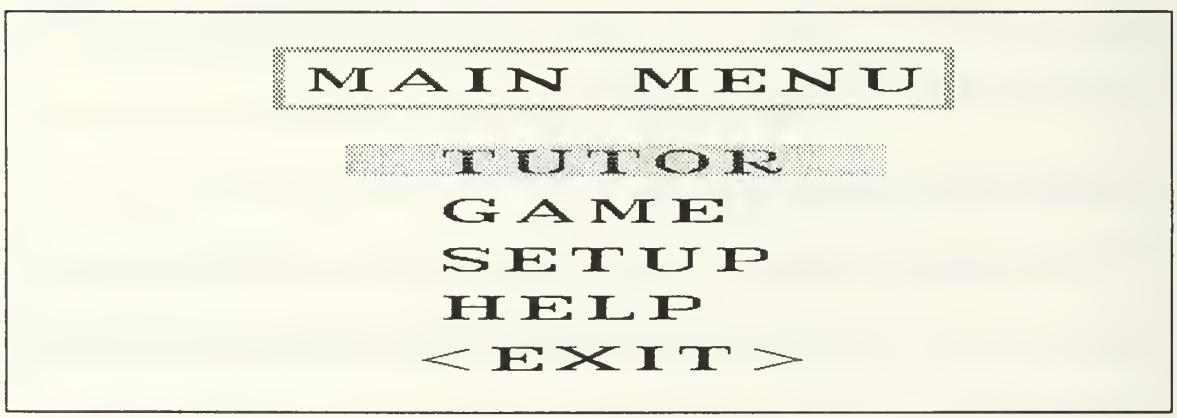



**Figure 4.3 Main Menu Display**

Once the new user has entered the tutor mode, the system diagnoses the user's level of ability in helicopter recognition based on the WETFUR features. Figure 4.4 shows a typical screen display for the diagnosis.

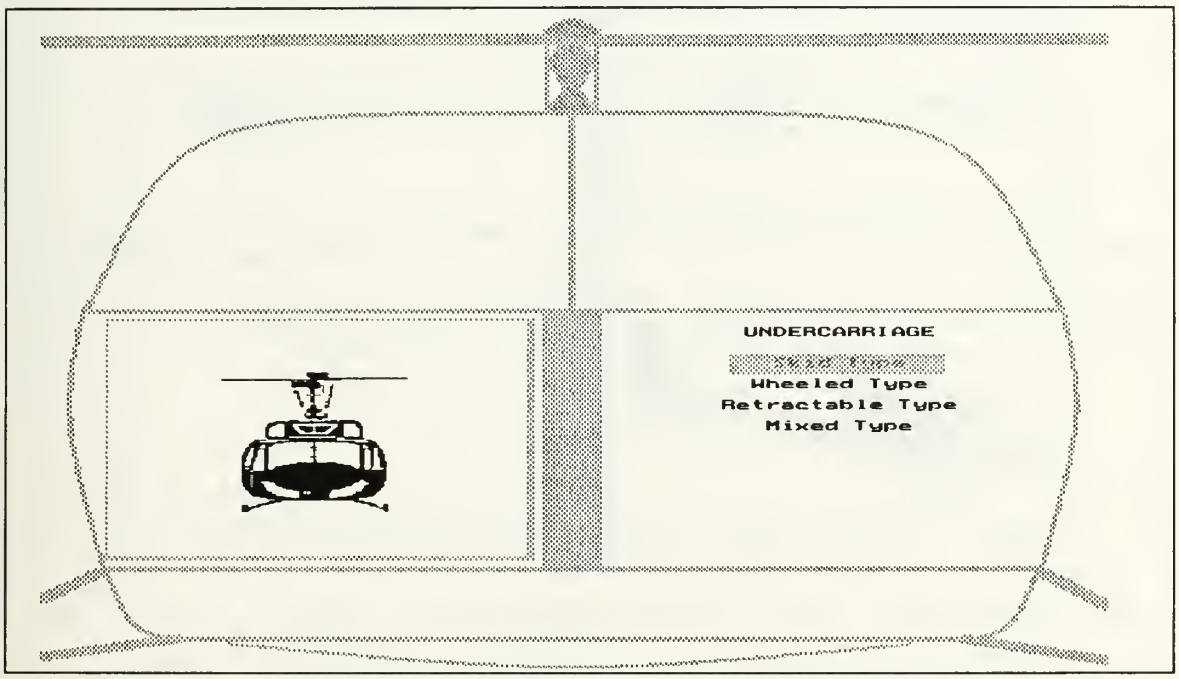



**Figure 4.4 The Diagnosis Display**

After the system has finished diagnosing the user's ability, the system determines which level is appropriate for the user. For example, if the user is in the novice level then the system begins in the teach mode. In the teach mode, the system presents a particular helicopter image and its WETFUR features to the user. Figure 4.5 shows a typical screen display for a user in the novice level and in the teach mode. The user can press the <Esc> key to quit any time during the process.

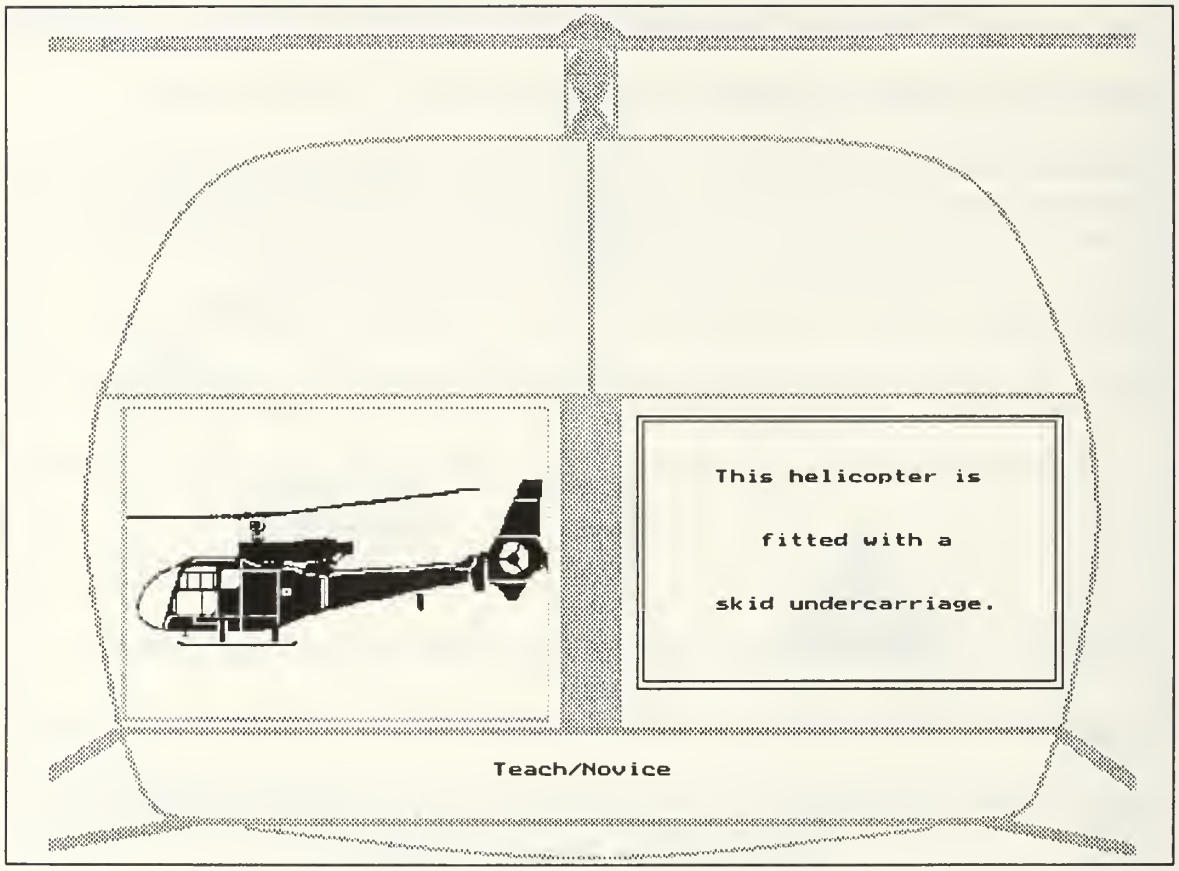



**Figure 4.5 The Teach/Novice Display**

The above descriptions simply describes one of the many purposes of the tutor system. The following figures depict other purposes that the system may fulfill. Figure 4.6 shows a typical screen display of the one-player game mode. Figure 4.7 shows a sample "Help" screen display.

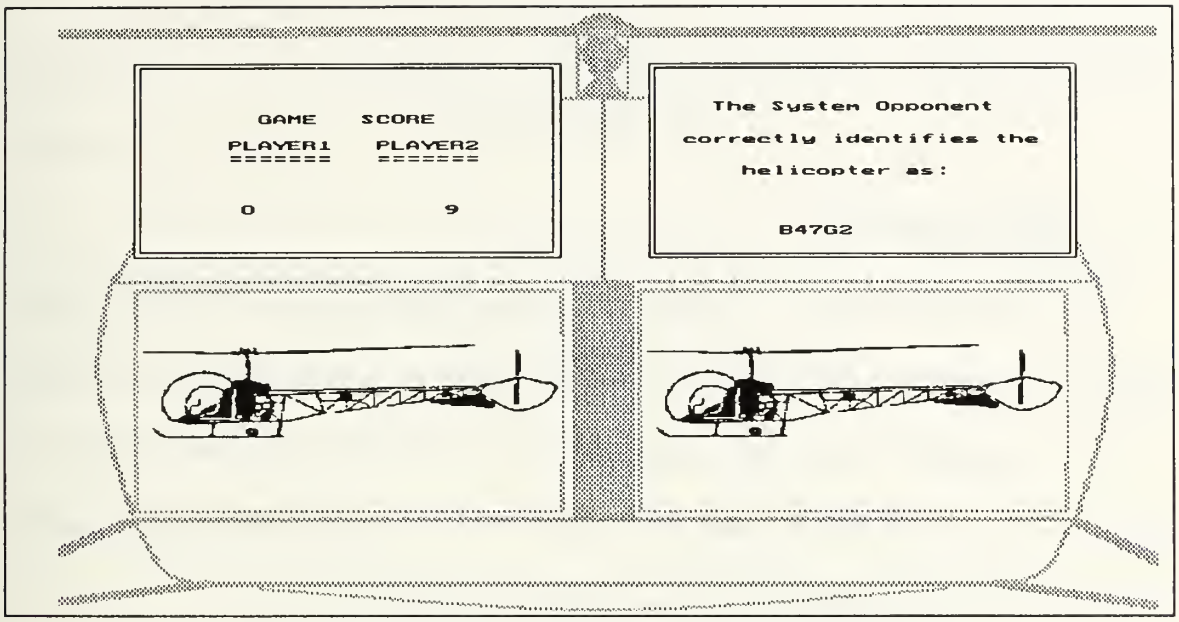



Figure 4.6 The One-Player Game Display

HELP! for the Helicopter Recognition Tutor

The Helicopter Recognition Tutor has two types of HELP! which are available.

General HELP! is available through the HELP! Menu and includes information on the following topics:

- ABOUT HELP - this HELP! screen.
- TUTOR HELP - overview of the TUTOR system.
- GAME HELP - overview of the 1 or 2 Player Game.
- SETUP/UTILITY HELP - description of the various Utilities available.

Context Sensitive HELP! is available from within the system by entering 'H' at any time, and provides detail HELP! based on the specific Mode/Level the user was in at the time of the HELP! resquest.

Enter any key to the HELP! MENU.

Figure 4.7 A sample Help Display

# V. CONCLUSION

## A. ACHIEVEMENTS

We have shown that it is possible to develop an ICAI system for tutoring visual helicopter recognition. The Helicopter Recognition Tutor is a portable program that can be run on any IBM compatible personal computer with VGA graphics adapter. In addition, it was developed by using the Object Oriented Programming (OOP) concept. Therefore, the system is modular, easy to maintain and reuse.

In order to help students to recognize helicopter, we developed the WETFUR theory for helicopter recognition and extracted many WETFUR features of helicopters from the references [Refs. 9, 10]. We believe that the WETFUR theory is helpful and useful for teaching helicopter recognition in the Helicopter Recognition Tutor.

## B. RECOMMENDATIONS

There are several areas that could be enhanced to improve the Helicopter Recognition Tutor. First, the user interface could be improved to:

- provide mouse input instead of keyboard input,
- make the screen more user friendly,
- allow a natural language input capability.

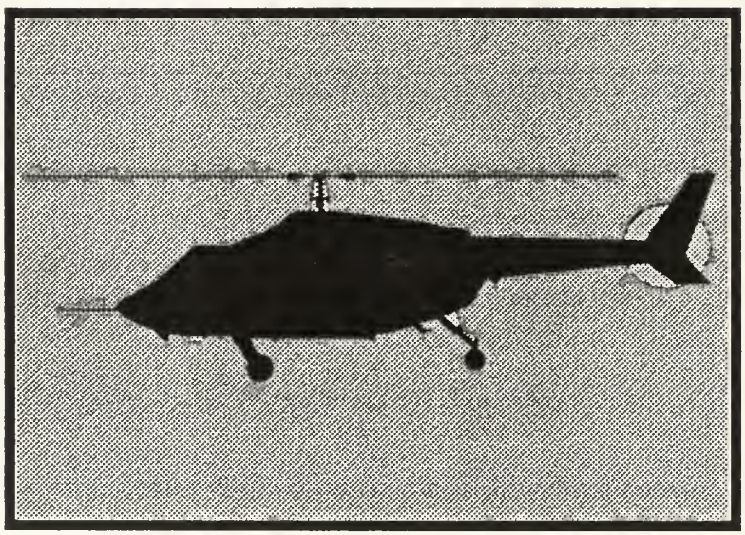
Second, the domain knowledge base must be increased to make the Helicopter Recognition a practical training tool. The system, as developed, contains only a limited number of helicopters.

# APPENDIX A - USER'S MANUAL

USER'S MANUAL

FOR

HELICOPTER RECOGNITION TUTOR



500313

## USER'S MANUAL

## TABLE OF CONTENTS

# I. INTRODUCTION

Helicopter recognition training is required for any soldier in the military. It is often taught by a military instructor using slides, photos, drawings and other training methods.

The purpose of developing this Helicopter Recognition Tutor is to show that we could use existing AI (Artificial Intelligent) techniques and technology in computer science to develop a computer-based training system. This tutor system is designed to identify the soldier's current ability in recognizing helicopters and teach them at a level suitable to that ability.

How to teach the soldier to identify a helicopter is based on the WETFUR theory (Wing, Engine, Tail, Fuselage, Undercarriage, and Rotor). It is of use to introduce visual helicopter recognition to new soldiers and give refresher training to more advanced soldiers, since this tutor is developed to identify the soldiers current ability and teach at a level appropriate to that ability.

## II. INSTALLATION

This chapter tells you how to install the Helicopter Recognition Tutor on your PC compatible computer. This tutor requires a PC compatible equipped with the following features:

- A hard disk with at least 6 MB free and one 3.5" floppy disk drive.
- A minimum of 512K of memory installed on your system.
- Dos version 2.1 or higher.
- VGA or super VGA.

The following steps tell you how to install the Helicopter Recognition Tutor on your own computer.

1. Power on your computer, then type C: and place the install disk (disk 1) in drive A or B (which depends on where your 3.5" is placed). Type A: or B: and press Enter.
2. Type INSTALL and press Enter. Tutor's INSTALL program will lead you through the installation procedures.
3. Follow the installation procedures shown on your screen.

As soon as the installation has completed, you can run the Tutor by typing HPT in the HP subdirectory.

## III. USING THE TUTOR

The tutor mode of the tutor system is composed of three levels: Novice, Intermediate, and Expert, and three tutoring modes: Teach, Review, and Test. Figure 3.1 depicts the structure of the tutor mode.

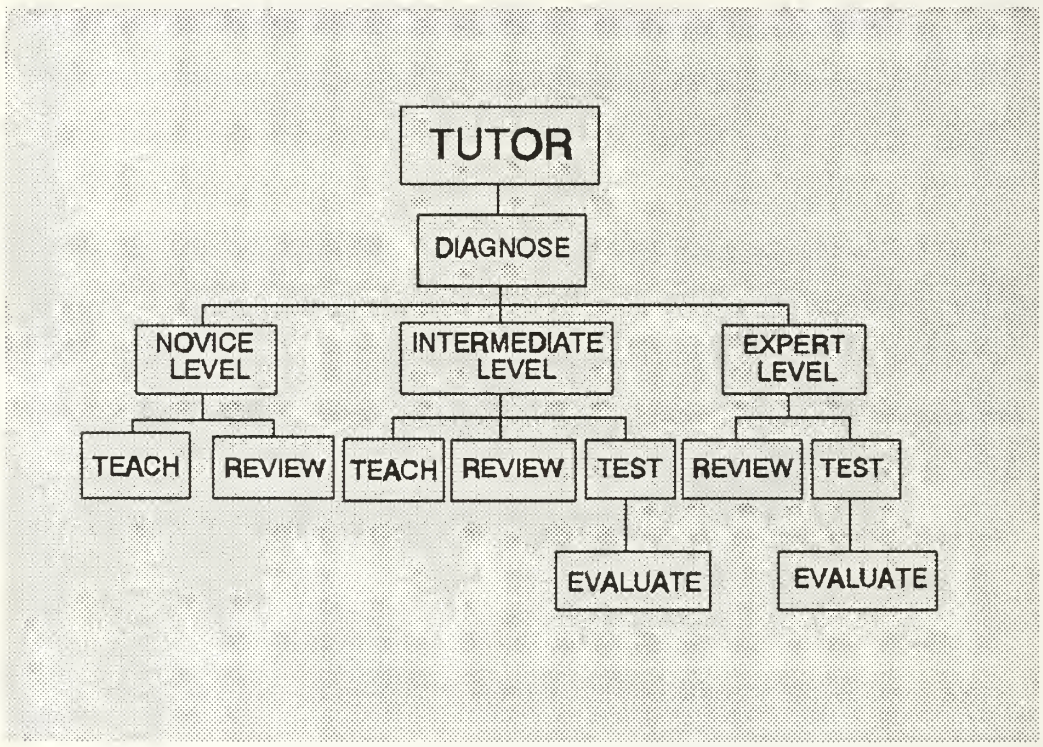



**Figure 3.1 Structure of the Tutor**

When a student wants to use the tutor mode, the student has to select the "Tutor" item from the Main Menu. The system will ask the user to enter an individual student's ID. If a new student is encountered, the system asks for the student's name. The system then diagnoses the student's level of ability at visual helicopter recognition by presenting ten helicopter to the student. The student is asked to recognize each helicopter based on the WETFUR features. If the student incorrectly responds to 2 or more of the 10 helicopters, he will begin at the

Novice Level. If the student correctly responds to 9 or 10, he will begin at the Intermediate Level.

### A. Novice Level

The Novice Level is for students that are new to visual helicopter recognition and have not mastered the WETFUR features of helicopters. The teach mode and the review mode are available in this level. In the teach mode, the student are taught the WETFUR features. In the review mode, WETFUR features are presented randomly. The system then asks the student to identify the WETFUR features or helicopter, and takes actions based on the student's response.

### B. Intermediate Level

The Intermediate Level is for students that have a capability to recognize WETFUR features of helicopter. In this level, students learn to identify specific helicopters visually based on their WETFUR features. There are three tutoring modes in this level: teach, review, and test. In the teach mode, students are taught the WETFUR features visible on specific helicopter. In the review mode, each helicopter is presented on the screen. The system then asks the student to identify the WETFUR features or helicopter, and takes actions based on the student's response. In the test mode, the student is presented with each helicopter. The system then asks the student to identify the helicopter, and keeps a record of the student's performance. If the student's performance is poor, the student may be forced to revert to a lower level.

### C. Expert Level

The Expert Level is for students that have mastered the WETFUR features of helicopters very well. In this level, there are two tutoring modes: review and test. In these two modes, the system does not present any helicopter images to students. Students have to identify a helicopter based on a description of the WETFUR features that the helicopter has. In addition, the student's performance will be maintained by the system. If the student's performance is poor, the student will be forced to revert to a lower level.

### D. Some Useful Keys

In the tutor system, some useful keys have a special purpose. They are described as follows:

| Press | to |
|---|---|
| <Enter> | Select the item that is highlighted, if a menu exists; Cause the program to continue after it has paused, if a menu doesn't exit. |
| <Esc> | Quit what you are doing. |
| C | Continue with the next tutor session, if the current tutor session is finished. |
| H | Get a help message. |
| O | Allow Player 1 to identify helicopter, when the system is in the one-player game mode. |
| T | Allow Player 2 to identify helicopter, when the system is in the two-player game mode. |

| | |
|---|---|
| Up Arrow Key | Highlight the item above the one currently highlighted in a menu. |
| Down Arrow Key | Highlight the item below the one currently highlighted in a menu. |
| PgUp | Cause the previous 8 selections to be shown in a menu. |
| PgDn | Cause the next 8 selections to be shown in a menu. |
| + | Scroll through the WETFUR features in forward order, when the system is in the expert level. |
| -- | Scroll through the WETFUR features in reverse order, when the system is in the expert level. |

## IV. PLAYING THE GAME

There are two game modes in the tutor system. One is the one-player game mode, the other is the two-player game mode.

In the one-player game mode, a student plays against the system. A helicopter will appear on two sides of the screen panel, when the game begins. If the player wants to identify the helicopter, he must press the "O" key. The player is then given a chance to identify the helicopter in the menu that will appear. Time for identifying the helicopter is limited. If the user does not enter the answer within the time limit, the system opponent will display the answer on the screen. The game will show 25 helicopters. The winner is the player who gets the highest score.

In the two-player game mode, two players compete against each other. The key for Player1 is the "O" key, and the key for Player2 is the "T" key. 25 helicopters are displayed during the game. The winner is the player who gets the highest score.

# V. USING UTILITIES

The Setup mode of this tutor system provides the various utilities for the system administrator to manage the tutor system and maintain the database. When a student wants to use the various utilities, the student have to select the "Setup" item from the Main Menu. The system then requests the student to enter a password. Once the password is entered correctly, the Setup Menus appears on the screen. If the password is entered incorrectly, the system returns to the Main Menu. The password can be found on the cover of the use's manual.

The Setup mode of this tutor system serves three purposes: (1) selecting the helicopter, (2) adding or modifying the helicopter, and (3) getting a student report. They will be described in the following sections.

## A. Selecting the Helicopter

The "Select Helicopter" utility allows the system administrator to select the helicopter that will be taught. The default list of helicopters is in Appendix C of this manual. Select the helicopter that is to be included in the system from the menu. Continue the selection process until all of the helicopters that will be taught have been selected. When you have finished, press <Esc> to return to the Setup Menu. In order to reach a high quality of training, we recommend that you select all of the helicopters that you want to be included in the system.

## B. Adding or Modifying Helicopter

If you want to add a new helicopter to the system, please use a scanner to scan three views (front, side, and bottom) of the helicopter from ***Jane's World Aircraft Recognition Handbook***. When it is completed, please do the following steps to add or modify helicopter.

1. Select the "Add/Modify Helicopter" item from the Setup menu, and press <Enter>.
2. Two dialogue boxes are displayed on the screen. They remind you to create the helicopter images, before you use this utility.
3. When the helicopter name is requested, type in the name of the helicopter and press <Enter>.
4. If the helicopter already exists, A menu will appear. The menu contains selections for a particular WETFUR feature. If not, a dialogue box appears on the screen and tells you the helicopter's image files are not present on the disk.
5. If the helicopter image files are found, then highlight the appropriate feature visible in this view of the helicopter. If the feature is not visible in this view of the helicopter, press <Esc>.
6. Repeat step 4 for each of the WETFUR feature menus.
7. Repeat step 4 for the other two views of the helicopter.

Note: The tutor system identifies the helicopter based on the first five characters in the name. If you define a new helicopter, make sure the given name is unique.

### D. Getting a Student Report

A student report contains the following information about a student:

- Student's ID.
- Student's name.
- Current mode that the student is in.
- Current level that the student is in.
- Last test score the student received.

This report can be used to track the performance of the student. To get a student report, the following steps are required:

1. Select the "Student Report" item from the Setup Menu, and press <Enter>.
2. Select the student's ID from the menu, and press <Enter>.
3. Repeat for other students as desired. When you have finished, press <Esc> to return to the Setup Menu.

# VI. GETTING HELP

The tutor provides two modes of getting help information. One is getting help from the Main Menu by selecting the "Help" item. This menu selection causes the Help Menu to show up. The Help Menu has the following topics for the user to select.

- About Help: Information about the Help available in the tutor system.
- Tutor Help: Overview of the tutor session.
- Game Help: Overview of the one or two player game.
- Setup/Utility Help: A description of the various utilities available in the tutor system.

The other mode is the context sensitive help. To get a context sensitive help, the user has to press the "H" key. This Help mode provides detailed information about the current mode/level that the user was in at the time of the Help request.

## APPENDIX B - SOURCE CODE

```
{$M $4000,0,48000}

program Install;

uses DOS, CRT, GRAPH;

var
   Ch       : char;
   grDriver : integer;
   grMode   : integer;
   ErrCode  : integer;
   DirInfo  : SearchRec;
   Disk1    : string[1];
   Disk2    : string[1];
begin
   grDriver := VGA;
   grMode := VGAHi;
   InitGraph(grDriver,grMode,'');
   ErrCode := GraphResult;
   if ErrCode <> 0 then
      begin
         writeln('This program requires VGA graphics.');
         writeln('          Install ABORTED.');
         writeln('  Press any key to return to DOS.');
         Ch := ReadKey;
         Exit;
      end;
   Disk1 := '';
   Disk2 := '';
   while (Disk1 <> 'A') and (Disk1 <> 'a') and (Disk1 <> 'B') and (Disk1 <> 'b')
do
   begin
     ClearDevice;
     SetColor(14);
     SetTextJustify(CenterText,CenterText);
```

```
   OutTextXY(320,100,'*****   Which driver is a source disk driver(A/B)
                ===>>');
   Ch := #8;
   SetWriteMode(CopyPut);
   while (Ch <> #13) do
   begin
      Ch := ReadKey;
      Disk1 := Concat(Disk1,Ch);
   end;
end;

while (Disk2 <> 'C') and (Disk2 <> 'c') and
                        (Disk2 <> 'D') and (Disk2 <> 'd') do
begin
   ClearDevice;
   SetColor(14);
   SetTextJustify(CenterText,CenterText);
   OutTextXY(320,100,'*****   Which Disk do you want to install(C/D)  ===>>');
   Ch := #8;
   SetWriteMode(CopyPut);
   while (Ch <> #13) do
   begin
      Ch := ReadKey;
      Disk2 := Concat(Disk2,Ch);
   end;
end;

ClearDevice;
OutTextXY(320,100,'Welcome to the Helicopter Recognition Tutor Install
            Program');
OutTextXY(300,250,'Press any key to start');
Ch := ReadKey;
ClearDevice;
OutTextXY(320,100,'Creating a new directory called ''HP''.');
ChDir(Concat(Disk2,':'));
MkDir('HP');
if IOResult <> 0 then
    begin
       OutTextXY(320,110,'Cannot create directory.  Install ABORTED.');
       OutTextXY(320,160,'Press any key to return to DOS.');
       Ch := ReadKey;
       ClearDevice;
       CloseGraph;
```

```
        Exit;
    end;
ChDir('HP');

ClearDevice;
FindFirst(Concat(Disk1,':','\Disk.1'),AnyFile,DirInfo);
while DOSError <> 0 do
    begin
        Sound(440);
        Delay(100);
        NoSound;
        ClearDevice;
        OutTextXY(320,100,Concat('Make sure that Disk 1 is in Drive ',Disk1,':
                    and press any key.'));
        Ch := ReadKey;
        FindFirst(Concat(Disk1,':','\Disk.1'),AnyFile,DirInfo);
    end;

ClearDevice;
OutTextXY(320,100,'Copying files from Disk 1');
ClearDevice;
SwapVectors;
Exec('C:\COMMAND.COM',Concat('/C copy ',Disk1,':\*.*'));
Exec('C:\COMMAND.COM',Concat('/C copy ',Disk1,':\DAT\*.*'));
Exec('C:\COMMAND.COM',Concat('/C copy ',Disk1,':\HLP\*.*'));
Exec('C:\COMMAND.COM',Concat('/C copy ',Disk1,':\MNU\*.*'));
Exec('C:\COMMAND.COM',Concat('/C copy ',Disk1,':\DEF\*.*'));
Exec('C:\COMMAND.COM',Concat('/C copy ',Disk1,':\REC\*.*'));
Exec('C:\COMMAND.COM',Concat('/C copy ',Disk1,':\NAM\*.*'));

SwapVectors;
ClearDevice;
OutTextXY(320,100,'Insert Disk 2 in Drive B: and press any key.');
Ch := ReadKey;
FindFirst(Concat(Disk1,':','\Disk.2'),AnyFile,DirInfo);
while DOSError <> 0 do
    begin
        Sound(440);
        Delay(100);
        NoSound;
        ClearDevice;
        OutTextXY(320,100,'Make sure that Disk 2 is in Drive B: and press any key.');
        Ch := ReadKey;
```

```
            FindFirst(Concat(Disk1,':','\Disk.2'),AnyFile,DirInfo);
        end;
    ClearDevice;
    OutTextXY(320,110,'Copying files from Disk 2');
    ClearDevice;
    SwapVectors;
    Exec('C:\COMMAND.COM',Concat('/C copy ',Disk1,':\*.*'));
    Exec('C:\COMMAND.COM',Concat('/C copy ',Disk1,':\PCX\*.*'));
    Exec('C:\COMMAND.COM',Concat('/C copy ',Disk1,':\MSG\*.*'));

    SwapVectors;
    ClearDevice;
    OutTextXY(320,100,'Insert Disk 3 in Drive B: and press any key.');
    Ch := ReadKey;
    FindFirst(Concat(Disk1,':','\Disk.3'),AnyFile,DirInfo);

    while DOSError <> 0 do
       begin
          Sound(440);
          Delay(100);
          NoSound;
          ClearDevice;
          OutTextXY(320,110,'Make sure that Disk 3 is in Drive B: and press any
key.');
          Ch := ReadKey;
          FindFirst(Concat(Disk1,':','\Disk.3'),AnyFile,DirInfo);
       end;

    ClearDevice;
    OutTextXY(320,100,'Copying files from Disk 3');
    ClearDevice;
    SwapVectors;
    Exec('C:\COMMAND.COM',Concat('/C copy ',Disk1,':\*.*'));
    Exec('C:\COMMAND.COM',Concat('/C copy ',Disk1,':\MSG\*.*'));
    Exec('C:\COMMAND.COM','/C del install.exe');
    Exec('C:\COMMAND.COM','/C del disk.*');

    SwapVectors;
    ClearDevice;
    OutTextXY(320,100,'Install program complete.');
    OutTextXY(310,160,'Press any key to return to DOS');
    Ch := ReadKey;
    ClearDevice;
```

```
    CloseGraph;
end.
```

```
{$M $4000,0,480000}

program Hpt;

uses CRT, Graph, GX_TP, PCX_TP, HpScreen, HpDialog, Menus, Mmenus,
     HpTutor,Game, Utility, Students, Helpshow;

type
    name = string[20];

const
    pcxtype = gxVGA_12;
    pcximage1 = 'Cover.pcx';
    pcximage2 = 'Menu.pcx';
    pcximage3 = 'Welcome.pcx';
    pcximage4 = 'Fame.pcx';


var
    HelpSh   : Hshow;
    Mainmenu : Mmenu;
    Helpmenu : Mmenu;
    GraphDriver, GraphMode, ErrorCode, ChoiceNum : Integer;
    Choice : name;
    Ch : char;
    F : text;
    FileName : string;
    Key : char;
    retcode : integer;
    Tempret : integer;

procedure HallofFame;
    var FameName : name;
    X,Y : integer;

begin
    if (retcode = gxSUCCESS) then begin
      retcode := pcxFileDisplay(pcximage4,0,0,0);
    end;
    Assign(F,'HallFame.rec');
    Reset(F);
```

```
    SetBkColor(Black);
    SetColor(10);
    X := 125;
    Y := 140;
    while (not eof(F)) and (X < 600) do
        begin
            Readln(F,FameName);
            OutTextXY(X,Y,FameName);
            Y := Y + 20;
            if Y > 350 then
                begin
                    X := X + 250;
                    Y := 140;
                end;
        end;
    Ch := ReadKey;
    SetColor(0);
    ClearDevice;
    Close(F);
end;{HallofFame}

procedure RunTutor;
begin
    {Initialize Graphics by PCX_utility}
    retcode := gxSetDisplay(pcxtype);
    retcode := gxSetMode(gxGRAPHICS);
    if (retcode = gxSUCCESS) then begin
      retcode := pcxFileDisplay(pcximage2,0,0,0);
    end;
    Readln;
    Tutor;
end; {RunTutor}

procedure RunGame;
begin
    if (retcode = gxSUCCESS) then begin
      retcode := pcxFileDisplay(pcximage2,0,0,0);
    end;
    PlayGame;
end; {RunGame}

procedure GetHelp;
begin
```

```
    Helpmenu.MShowMenu('helpmenu.pcx');
    Choice := Helpmenu.MGetChoice;
    while (Choice <> 'EXIT HELP!') and (Choice <> 'null') do
        begin
            if Choice = 'ABOUT HELP!' then
                Helpsh.screen('help.hlp')
            else if Choice = 'TUTOR HELP!' then
                Helpsh.screen('tutor.hlp')
            else if Choice = 'GAME HELP!' then
                Helpsh.screen('game.hlp')
            else if Choice = 'SETUP/UTILITY HELP!' then
                Helpsh.screen('setup.hlp');
            GotoXY(1,1);
            Helpmenu.MInit('Help.mnu');
            Helpmenu.MShowMenu('helpmenu.pcx');
            Choice := HelpMenu.MGetChoice;
        end;
    Mainmenu.MInit('Main.mnu');
    Mainmenu.MShowMenu('Mainmenu.pcx');
end; {GetHelp}

{Main Program}
begin
    {Initialize Graphics by PCX_utility}
    retcode := gxSetDisplay(pcxtype);
    retcode := gxSetMode(gxGRAPHICS);
    if (retcode = gxSUCCESS) then begin
      retcode := pcxFileDisplay(pcximage1,0,0,0);
      Key := Readkey;
      retcode := pcxFileDisplay(pcximage2,0,0,0);
      retcode := pcxFileDisplay(pcximage3,120,105,0);
      retcode := pcxFileDisplay(pcximage3,340,105,0);
      Key := ReadKey;
      Tempret := gxSetMode(gxTEXT);
    end;
    {Initialize Graphics Adapter to VGA 640x480 16-color mode}
    GraphDriver := VGA;
    GraphMode := VGAHi;
    InitGraph(GraphDriver, GraphMode, '');
    SetBkColor(Black);
    {Load the graphics and data into memory}
    Mainmenu.MInit('Main.mnu');
    Helpmenu.MInit('Help.mnu');
```

```
    HallofFame;
    {Display the Initial Menu Screen and Get a Response}
    StudentModel.Mode := '';
    Mainmenu.MShowMenu('Mainmenu.pcx');
    Choice := Mainmenu.MGetChoice;
    while (Choice <> 'EXIT') and (Choice <> 'null') do
       begin
          if Choice = 'TUTOR SESSION' then RunTutor
          else if Choice = 'GAME' then RunGame
          else if Choice = 'HELP!' then GetHelp
          else if Choice = 'SETUP' then SetUp;
          GotoXY(1,1);
          StudentModel.Mode := '';
          Mainmenu.MInit('Main.mnu');
          Helpmenu.MInit('Help.mnu');
          Mainmenu.Mshowmenu('Mainmenu.pcx');
          Choice := Mainmenu.MGetChoice;
       end;
    CloseGraph;
end. {Main Program}
```

```
unit HpScreen;

interface

uses Graph;

type
  Screen = object
    X, Y : Integer;
    F : File;
    MemSize : Word;
    P : Pointer;
    IsVisible : Boolean;
    constructor Init(FileName : String);
    procedure Show(XLoc, YLoc : Integer);
    procedure Hide;
    destructor Kill;
  end;

implementation

constructor Screen.Init(FileName : String);
begin
  IsVisible := False;
  Assign(F, FileName);          {Prepare the file                    }
  Reset(F, 1);                  {for a read operation.               }
  MemSize := FileSize(F);       {Determine memory needed             }
  GetMem(P, MemSize);           {and allocate the memory on the heap. }
  BlockRead(F, P^, MemSize); {Read in the graphic pic file       }
  Close(F);                     {and close the file.                 }
end;

procedure Screen.Show(XLoc, YLoc : Integer);
begin
  if not IsVisible then
    begin
      X := XLoc;
      Y := YLoc;
      PutImage(X,Y,P^,CopyPut);  {Display the graphics on the screen.}
      IsVisible := True;
    end;
end;
```

```
procedure Screen.Hide;
begin
  if IsVisible then
    begin
      PutImage(X,Y,P^,XorPut);   {Hide all pixels.}
       IsVisible := False;
    end;
end;

destructor Screen.Kill;
begin
  FreeMem(P, MemSize);   {Release the heap memory.}
end;

end.
```

```
unit HpDialog;

interface

uses Graph, HpScreen;

type
  Dialog = object(Screen)
    OldP : Pointer;
    DOldP : Pointer;
    procedure Show(XLoc, YLoc : integer);
    procedure Hide;
  end;

implementation

procedure Dialog.Show(XLoc, YLoc : integer);
begin
  if not IsVisible then
    begin
      X := XLoc;
      Y := YLoc;
      GetMem(OldP, MemSize);              {Save the old graphics     }
      GetImage(X,Y,X+230,Y+150,OldP^);
      PutImage(X,Y,P^,CopyPut);           {and dispaly the new one. }
      IsVisible := True;
    end;
end;

procedure Dialog.Hide;
begin
  if IsVisible then
    begin
      PutImage(X,Y,OldP^,CopyPut);     {Restore the old graphics       }
      FreeMem(OldP, MemSize);          {and release the heap memory. }
      IsVisible := False;
     end;
end;

end.
```

```
unit Helicopt;

interface

uses Graph, PCX_TP, GX_TP, HpScreen;

type
  Name = String[20];

  HPTData = record
    HelicopterName : Name;
    ExampleOf : Name;
    ExampleInfo : Name;
    Wings : array [1..1] of Name;
    WingsInfo : array [1..1] of Name;
    Engine : array [1..1] of Name;
    EngineInfo : array [1..1] of Name;
    Fuslag : array [1..1] of Name;
    FuslagInfo : array [1..1] of Name;
    Trot : array [1..1] of Name;
    TrotInfo : array [1..1] of Name;
    Mrot : array [1..1] of Name;
    MrotInfo : array [1..1] of Name;
    Ucag : array [1..1] of Name;
    UcagInfo : array [1..1] of Name;
    Hstn : array [1..1] of Name;
    HstnInfo : array [1..1] of Name;
    Hstl : array [1..1] of Name;
    HstlInfo : array [1..1] of Name;
  end;

  Helicopter= object(Screen)
    Aptr : Pointer;
    Ap : Pointer;
    Msize : Word;
    F1 : Text;
    HPTInfo : HPTData;
    constructor Init(HPTName : Name);
    procedure Show(XLoc,YLoc : integer; HPTName : Name);
    procedure Hide;
    procedure Kill;
  end;
```

```
var
  retcode : Integer;

implementation

constructor Helicopter.Init(HPTName : Name);
var
  Counter : Integer;
begin
  IsVisible := False;
  Assign(F1,Concat(HPTName,'.dat'));
  Reset(F1);
  Readln(F1,HPTInfo.HelicopterName);
  Readln(F1,HPTInfo.ExampleOf);
  Readln(F1,HPTInfo.ExampleInfo);
  for Counter := 1 to 1 do
    begin
      Readln(F1,HPTInfo.Wings[Counter]);
      Readln(F1,HPTInfo.WingsInfo[Counter]);
    end;
  for Counter := 1 to 1 do
    begin
      Readln(F1,HPTInfo.Engine[Counter]);
      Readln(F1,HPTInfo.EngineInfo[Counter]);
    end;
  for Counter := 1 to 1 do
    begin
      Readln(F1,HPTInfo.Fuslag[Counter]);
      Readln(F1,HPTInfo.FuslagInfo[Counter]);
    end;
  for Counter := 1 to 1 do
    begin
      Readln(F1,HPTInfo.Trot[Counter]);
      Readln(F1,HPTInfo.TrotInfo[Counter]);
    end;
  for Counter := 1 to 1 do
    begin
      Readln(F1,HPTInfo.Mrot[Counter]);
      Readln(F1,HPTInfo.MrotInfo[Counter]);
    end;
  for Counter := 1 to 1 do
    begin
      Readln(F1,HPTInfo.Ucag[Counter]);
```

```
        Readln(F1,HPTInfo.UcagInfo[Counter]);
      end;
  for Counter := 1 to 1 do
      begin
        Readln(F1,HPTInfo.Hstn[Counter]);
        Readln(F1,HPTInfo.HstnInfo[Counter]);
      end;
  for Counter := 1 to 1 do
      begin
        Readln(F1,HPTInfo.Hstl[Counter]);
        Readln(F1,HPTInfo.HstlInfo[Counter]);
      end;
  Close(F1);
end;

procedure Helicopter.Show(XLoc, YLoc : integer; HPTName : Name);
begin
  if not IsVisible then
    begin
      retcode := gxSetDisplay(gxVGA_12);
      X := XLoc;
      Y := YLoc;
      retcode := pcxFileDisplay(concat(HPTName,'.pcx'),X,Y,0);
      Msize := ImageSize(0,0,234,175);
      GetMem(Aptr,Msize);
      GetImage(X,Y,234+X,175+Y,Aptr^);
      IsVisible := True;

    end;
end;

procedure Helicopter.Hide;
begin
  if IsVisible then
    begin
      PutImage(X,Y,Aptr^,XorPut);
      IsVisible := False;
    end;
end;

procedure Helicopter.Kill;
begin
  IsVisible := False;
```

```
   Freemem(Aptr,Msize);
end;

end.
```

```
unit Students;

interface

type
  Name = string [20];

  Model = object
    StudentName, Mode, Level : Name;
    TestScore : Integer;
    NumShown, NumMissed : Integer;
    HPTArray : array [1..150] of Name;
    MissedArray : array [1..150] of Name;
    function Get : Boolean;

    procedure Update(StuName:Name;NewMode:Name;NewLevel:Name;
                     NewScore:Integer);
    procedure Save;
    function GetEntry(MaxNum : Integer) : Integer;
    function AddEntry(HPTName : Name; MaxNum :Integer) : Boolean;
    procedure Kill;
  end;

var
  StudentModel : Model;

implementation

uses DOS, CRT, Graph, HpDialog;

var
  FileName : Name;
  Deleted : Boolean;
  F : Text;
  S : Pathstr;
  Ch : Char;
  Counter : Integer;
  DialogScreen : Dialog;

function Model.GetEntry(MaxNum : Integer) : Integer;
begin
  Randomize;
  Counter := 1;
```

```
  while (StudentModel.HPTArray[Counter] = '') and (Counter <= MaxNum) do
    Counter := Counter + 1;
  if Counter < MaxNum then
    begin
      Counter := Random(MaxNum - 1);
      while StudentModel.HPTArray[Counter + 1] = '' do
        Counter := Random(MaxNum);
        GetEntry := Counter + 1;
    end
   else GetEntry := 0;
end;

function Model.AddEntry(HPTName : Name; MaxNum : Integer) : Boolean;
begin
  Counter := 1;
  while (StudentModel.MissedArray[Counter] <> '') and (Counter <= MaxNum+1) do
    Counter := Counter + 1;
  if Counter <= MaxNum then
    begin
      StudentModel.MissedArray[Counter] := HPTName;
      AddEntry := true;
    end
  else AddEntry := false;
end;

function Model.Get : Boolean;
const
   ALPHA  = ['A'..'Z','a'..'z'];
   NUM  = ['0'..'9'];
begin
   Deleted := False;
   FileName := '';
   Counter := 1;
   SetLineStyle(SolidLn,0,ThickWidth);
   SetWriteMode(CopyPut);
   SetColor(13);
   DialogScreen.Init('Model.msg');
   DialogScreen.Show(205,60);
   Line(280,165,350,165);
   Line(280,167,350,167);
   Line(280,169,350,169);
   Line(280,171,350,171);
   Line(280,173,350,173);
```

```
Line(280,175,350,175);
Line(280,177,350,177);
while Counter <= 6 do
  begin
    SetColor(10);
    Ch := ReadKey;
    if (Counter = 1) and (Ch in ALPHA) then
      begin
        OutTextXY(280+10*Counter,168,Ch);
        FileName := Concat(FileName,Ch);
        Counter := Counter + 1;
      end
    else if (Counter > 1) and (Counter < 6) and (Ch in NUM) then
      begin
        OutTextXY(280+10*Counter,168,Ch);
        FileName := Concat(FileName,Ch);
        Counter := Counter + 1;
      end
    else if (Ch = #8) and (Counter > 1) then
      begin
        SetWriteMode(CopyPut);
        SetColor(13);
        Line(260+10*Counter,165,290+10*Counter,165);
        Line(260+10*Counter,167,290+10*Counter,167);
        Line(260+10*Counter,169,290+10*Counter,169);
        Line(260+10*Counter,171,290+10*Counter,171);
        Line(260+10*Counter,173,290+10*Counter,173);
        Line(260+10*Counter,175,290+10*Counter,175);
        Line(260+10*Counter,177,290+10*Counter,177);
        Counter := Counter - 1;
        FileName := Copy(FileName,1,Counter-1);
        SetColor(13);
      end
    else if (Counter = 6) and (Ch = #13) then
        Counter := Counter + 1
      else
        begin
          Sound(440);
          Delay(100);
          NoSound;
        end;
  end;
DialogScreen.Hide;
```

```
    DialogScreen.Kill;
    S := FSearch('.',Concat(FileName,'.mdl'));
    if S = '' then
       begin
          Get := False;
          Exit;
       end
    else
       begin
          Assign(F,Concat(FileName,'.mdl'));
          Reset(F);
          Readln(F,StudentModel.StudentName);
          Readln(F,StudentModel.Mode);
          Readln(F,StudentModel.Level);
          Readln(F,StudentModel.TestScore);
          Readln(F,StudentModel.NumShown);
          Readln(F,StudentModel.NumMissed);
          for Counter := 1 to 150 do
             Readln(F,StudentModel.HPTArray[Counter]);
          for Counter := 1 to 150 do
             Readln(F,StudentModel.MissedArray[Counter]);
          Close(F);
       end;
    Get := true;
end; {GetStudentModel}

procedure Model.Update(StuName:Name;NewMode:Name;NewLevel:Name;
                       NewScore:Integer);
begin
   if NewLevel = 'Novice' then
      Assign(F,'hnovice.def')
   else
      Assign(F,'hintermd.def');
   Reset(F);
   Readln(F,StudentModel.StudentName);
   Readln(F,StudentModel.Mode);
   Readln(F,StudentModel.Level);
   Readln(F,StudentModel.TestScore);
   Readln(F,StudentModel.NumShown);
   Readln(F,StudentModel.NumMissed);
   SetColor(4);
   for Counter := 1 to 150 do
      begin
```

```
        Readln(F,StudentModel.HPTArray[Counter]);
      end;
   for Counter := 1 to 150 do
      Readln(F,StudentModel.MissedArray[Counter]);
   Close(F);
   StudentModel.StudentName := StuName;
   StudentModel.Mode := NewMode;
   StudentModel.Level := NewLevel;
   StudentModel.TestScore := NewScore;
end; {Update}

procedure Model.Save;
begin
   if not Deleted then
      begin
         Assign(F,Concat(FileName,'.mdl'));
         Rewrite(F);
         Writeln(F,StudentModel.StudentName);
         Writeln(F,StudentModel.Mode);
         Writeln(F,StudentModel.Level);
         Writeln(F,StudentModel.TestScore);
         Writeln(F,StudentModel.NumShown);
         Writeln(F,StudentModel.NumMissed);
         for Counter := 1 to 150 do
            Writeln(F,StudentModel.HPTArray[Counter]);
         for Counter := 1 to 150 do
            Writeln(F,StudentModel.MissedArray[Counter]);
         Close(F);
      end;
end;

procedure Model.Kill;
begin
   Exec('\COMMAND.COM',Concat('/C del ',FileName,'.mdl'));
   Deleted := true;
end;

end.
```

```
unit Helpshow;

interface

uses CRT, Graph, GX_TP, PCX_TP;

type
  Hshow = object
    constructor Screen(HFilename : String);
  end;

implementation

var
  Filename  : String[20];
  retcode   : Integer;
  IsVisible : Boolean;
  MemSize   : Word;
  Oldp1     : Pointer;
  Oldp2     : Pointer;
  Oldp3     : Pointer;
  Oldp4     : Pointer;
  Oldp5     : Pointer;
  Hptr      : Pointer;
  Ch        : Char;

constructor Hshow.Screen(HFilename : String);
begin
  retcode := gxSetDisplay(gxVGA_12);
  retcode := gxSetMode(gxGRAPHICS);
  IsVisible := False;
  if not IsVisible then
    begin
      Mark(Hptr);
      MemSize := 64000;
      GetMem(OldP1, MemSize);          {Save the old bitmap  }
      GetImage(0,0,639,099,OldP1^);
      GetMem(OldP2, Memsize);          {Save the old bitmap  }
      GetImage(0,100,639,199,OldP2^);
      GetMem(OldP3, MemSize);          {Save the old bitmap  }
      GetImage(0,200,639,299,OldP3^);
      GetMem(OldP4, MemSize);
```

```
        GetImage(0,300,639,399,OldP4^);
        MemSize := 51200;
        GetMem(OldP5, MemSize);
        GetImage(0,400,639,479,OldP5^);
        if (retcode = gxSUCCESS) then
          begin
            retcode := pcxFileDisplay(HFilename,0,0,0);
          end;
        IsVisible := true;
      end;
    Ch := ReadKey;
    if IsVisible then
      begin
        PutImage(0,0,OldP1^,CopyPut);      {Put the old bitmap back}
        MemSize := 64000;
        FreeMem(OldP1, MemSize);           {and free the heap memory.}
        PutImage(0,100,OldP2^,CopyPut);    {Put the old bitmap back  }
        MemSize := 64000;
        FreeMem(OldP2, MemSize);           {and free the heap memory.}
        PutImage(0,200,OldP3^,CopyPut);
        MemSize := 64000;
        FreeMem(OldP3, MemSize);
        PutImage(0,300,OldP4^,CopyPut);
        MemSize := 64000;
        freeMem(OldP4, MemSize);
        PutImage(0,400,OldP5^,CopyPut);    {Put the old bitmap back}
        MemSize := 51200;
        FreeMem(OldP5, MemSize);           {and free the heap memory.}
        Release(Hptr);
        IsVisible := False;
       end;
end;

end.
```

```
unit Help;

interface

uses CRT, Graph, HpScreen, Students, GX_TP, PCX_TP, Helpshow;

type
  HELPS = object
    constructor GetHelp;
  end;

implementation

var
  Ch : Char;
  FileName : Name;
  Showhelp : Hshow;

constructor Helps.GetHelp;
begin
  if StudentModel.Mode = '' then
    Showhelp.Screen('Menu.hlp')
  else if StudentModel.Mode = 'Game' then
    Showhelp.Screen('Game.hlp')
  else if StudentModel.Mode = 'Setup' then
    Showhelp.Screen('Setup.hlp')
  else if StudentModel.Mode = 'StuRep' then
    Showhelp.Screen('StuRep.hlp')
  else if StudentModel.Mode = 'SelectHPT' then
    Showhelp.Screen('SelHPT.hlp')
  else if StudentModel.Mode = 'AddHPT' then
    Showhelp.Screen('AddHPT.hlp')
  else if StudentModel.Mode = 'Diagnose' then
    Showhelp.Screen('Diagnose.hlp')
  else if StudentModel.Mode = 'Teach' then
    begin
      if StudentModel.Level = 'Novice' then
        Showhelp.Screen('TeaNov.hlp')
      else Showhelp.Screen('TeaInt.hlp');
    end
  else if StudentModel.Mode = 'Test' then
    begin
      if StudentModel.Level = 'Intermediate' then
```

```
            Showhelp.Screen('TestInt.hlp')
          else Showhelp.Screen('TestExp.hlp')
        end
      else
        begin
          if StudentModel.Level = 'Novice' then
            Showhelp.Screen('RevNov.hlp')
          else if StudentModel.Level = 'Intermediate' then
            Showhelp.Screen('RevInt.hlp')
          else Showhelp.Screen('RevExp.hlp');
        end;
end;

end.
```

```
unit Menus;

interface

uses CRT, Graph, Help, Students;

type
  Name = String[20];

  Menu = object
    MenuTitle : String;
    X,Y,Color,TopItem,SelectedItem,HighLightedItem : Integer;
    NumSelections : Integer;
    MenuSelection : array[1..150] of Name;
    F : Text;
    constructor Init(FileName : Name);
    function GetChoice : Name;
  end;

var
  LastSelection : Integer;
  HelpItem      : Helps;

implementation

const
  MaxSelections = 8;

var
  Ch : Char;
  Counter : Integer;
  Liner   : Integer;

constructor Menu.Init(FileName : Name);
begin
  HighlightedItem := 1;
  SelectedItem := 1;
  TopItem := 1;
  Counter := 0;
  Assign(F,FileName);
  Reset(F);
  Readln(F,MenuTitle);
  Readln(F,X);
```

```
   Readln(F,Y);
   Readln(F,Color);
   while (not eof(F)) and (Counter < 150) do
      begin
         Counter := Counter + 1;
         Readln(F,MenuSelection[Counter]);
      end;
   NumSelections := Counter;
   Close(F);
end;

function Menu.GetChoice : Name;

   procedure ShowMenu(Selection : Integer);
   begin
      SetColor(0);
      SetLineStyle(SolidLn,0,ThickWidth);
      SetWriteMode(CopyPut);
      for Liner := 1 to 18 do
          begin
             Line(391,227+Liner,560,227+Liner);
          end;
      SetTextJustify(CenterText,CenterText);
      SetColor(Color);
      OutTextXY(X,Y,MenuTitle);
      Counter := 0;
      while (Counter + Selection <= NumSelections) and
            (Counter + Selection < Selection + MaxSelections) do
        begin
          OutTextXY(X,Counter*15+Y+22,
          MenuSelection[Counter + Selection]);
          Counter := Counter + 1;
        end;
      SetColor(0);
      SetLineStyle(SolidLn,0,ThickWidth);
      SetWriteMode(CopyPut);
      for Liner := 1 to 18 do
          begin
             Line(X-75,Y+140+Liner,X+75,Y+140+Liner);
          end;
      SetColor(Color);
      if (Selection > 1) and (Selection <= NumSelections - 8) then
        OutTextXY(X,Y+150,'PgUp/PgDn')
```

```
   else if Selection > 1 then
      OutTextXY(X,Y+150,'PgUp')
   else if Selection <= NumSelections - 8 then
      OutTextXY(X,Y+150,'PgDn');
end; {ShowMenu}

procedure KillMenu;
begin
   if Color <> 0 then
      SetColor(0);
   SetWriteMode(CopyPut);
   Line(X-50,Y-3,X+50,Y-3);
   Line(X-50,Y,X+50,Y);
   Line(X-50,Y+3,X+50,Y+3);
   for Counter := 1 to 8 do
       begin
          Line(X-85,Counter*15+Y+3,X+85,Counter*15+Y+3);
          Line(X-85,Counter*15+Y+6,X+85,Counter*15+Y+6);
          Line(X-85,Counter*15+Y+9,X+85,Counter*15+Y+9);
       end;
   Line(X-35,Y+97,X+35,Y+97);
   Line(X-35,Y+100,X+35,Y+100);
   Line(X-35,Y+103,X+35,Y+103);
end; {KillMenu}

procedure Highlight(NewItem : Integer);
begin
   SetWriteMode(XorPut);
   SetColor(Color);
   Line(X-85,HighlightedItem*15+Y+3,X+85,HighlightedItem*15+Y+3);
   Line(X-85,HighlightedItem*15+Y+6,X+85,HighlightedItem*15+Y+6);
   Line(X-85,HighlightedItem*15+Y+9,X+85,HighlightedItem*15+Y+9);
   if HighLightedItem <> NewItem then
      begin
         HighlightedItem := NewItem;
         Line(X-85,NewItem*15+Y+3,X+85,NewItem*15+Y+3);
         Line(X-85,NewItem*15+Y+6,X+85,NewItem*15+Y+6);
         Line(X-85,NewItem*15+Y+9,X+85,NewItem*15+Y+9);
      end;
end; {Highlight}

procedure PageUp;
begin
```

```
   if TopItem > MaxSelections then
      begin
         KillMenu;
         SelectedItem := (SelectedItem - 8) - (HighlightedItem - 1);
         TopItem := TopItem - 8;
         HighlightedItem := 1;
         ShowMenu(TopItem);
         Highlight(HighlightedItem);
      end;
end; {PageUp}

procedure PageDown;
begin
   if TopItem + MaxSelections <= NumSelections then
      begin
         KillMenu;
         SelectedItem := (SelectedItem + 8) - (HighlightedItem - 1);
         TopItem := TopItem + 8;
         HighlightedItem := 1;
         ShowMenu(TopItem);
         Highlight(HighlightedItem);
      end;
end; {PageDown}

procedure MoveUp;
begin
   if HighlightedItem > 1 then
      begin
         Highlight(HighlightedItem - 1);
         SelectedItem := SelectedItem - 1;
      end;
end; {MoveUp}

procedure MoveDown;
begin
   if (HighlightedItem < MaxSelections) and
      (SelectedItem < NumSelections) then
      begin
         Highlight(HighlightedItem + 1);
         SelectedItem := SelectedItem + 1;
      end;
end; {MoveDown}
```

```
    procedure GetInput;
    begin
      Ch := ReadKey;
      begin
        Case Ch of
            'h'             : HelpItem.GetHelp;
            chr(80),chr(50) : MoveDown;
            chr(72),chr(56) : MoveUp;
            chr(81),chr(51) : PageDown;
            chr(73),chr(57) : PageUp;
        end;
      end;
    end; {GetInput}

begin
    ShowMenu(1);
    Highlight(1);
    repeat
      GetInput
    until (Ch = #13) or (Ch = #27);
    if Ch = #27 then
      GetChoice := 'null'
    else
      GetChoice := MenuSelection[SelectedItem];
    SetColor(0);
    SetLineStyle(SolidLn,0,ThickWidth);
    SetWriteMode(CopyPut);
    for Liner := 1 to 18 do
        begin
          Line(391,227+Liner,560,227+Liner);
        end;
    for Liner := 1 to 18 do
        begin
          Line(X-75,Y+140+Liner,X+75,Y+140+Liner);
        end;
    LastSelection := SelectedItem;
    KillMenu;
    TopItem := 1;
    SelectedItem := 1;
    HighlightedItem := 1;
end; {Menu.GetChoice}

end.
```

```
unit MMenus;

interface

uses CRT, Graph, Help, GX_TP, PCX_TP;

type
  Name = String[20];
  MMenu = object
    MTopItem,MSelectedItem,MHighLightedItem : Integer;
    MNumSelections : Integer;
    MMenuSelection : array[1..10] of Name;
    MF : Text;
    constructor MInit(MFileName : Name);
    procedure MShowMenu(MFileName : Name);
    function MGetChoice : Name;
  end;
var
  MLastSelection : Integer;
  retcode        : Integer;
  Helpitem       : helps;

implementation

const
  MMaxSelections = 5;

var
  MCh : Char;
  MCounter : Integer;

constructor Mmenu.MInit(MFileName : Name);
begin
  MHighlightedItem := 1;
  MSelectedItem := 1;
  MTopItem := 1;
  MCounter := 0;
  Assign(MF,MFileName);
  Reset(MF);
  while (not eof(MF)) and (MCounter < 10) do
    begin
      MCounter := MCounter + 1;
      Readln(MF,MMenuSelection[MCounter]);
```

```
      end;
   MNumSelections := MCounter;
   Close(MF);
end;

procedure Mmenu.MShowMenu(MFileName : Name);
begin
   retcode := gxSetDisplay(gxVGA_12);
   retcode := gxSetMode(gxGRAPHICS);
   if (retcode = gxSUCCESS) then
      begin
         retcode := pcxFileDisplay(MFileName,0,0,0);
      end;
   if (retcode <> gxSUCCESS) then
      begin
         writeln('An error occured: [',retcode,']');
      end;
end;

function Mmenu.MGetChoice : Name;

   procedure MHighlight(MNewItem : Integer);
   begin
      SetLineStyle(Solidln,0,ThickWidth);
      SetWriteMode(XorPut);
      SetColor(13);
      Line(220,MHighlightedItem*40+95+3,420,
      MHighlightedItem*40+95+3);
      Line(220,MHighlightedItem*40+95+6,420,
      MHighlightedItem*40+95+6);
      Line(220,MHighlightedItem*40+95+9,420,
      MHighlightedItem*40+95+9);
      Line(220,MHighlightedItem*40+95+12,420,
      MHighlightedItem*40+95+12);
      Line(220,MHighlightedItem*40+95+15,420,
      MHighlightedItem*40+95+15);
      Line(220,MHighlightedItem*40+95+18,420,
      MHighlightedItem*40+95+18);
      Line(220,MHighlightedItem*40+95+21,420,
      MHighlightedItem*40+95+21);
      Line(220,MHighlightedItem*40+95+24,420,
      MHighlightedItem*40+95+24);
      Line(220,MHighlightedItem*40+95+27,420,
```

```
   MHighlightedItem*40+95+27);
   if MHighLightedItem <> MNewItem then
      begin
         MHighlightedItem := MNewItem;
         Line(220,MHighlightedItem*40+95+3,420,
              MHighlightedItem*40+95+3);
         Line(220,MHighlightedItem*40+95+6,420,
              MHighlightedItem*40+95+6);
         Line(220,MHighlightedItem*40+95+9,420,
              MHighlightedItem*40+95+9);
         Line(220,MHighlightedItem*40+95+12,420,
              MHighlightedItem*40+95+12);
         Line(220,MHighlightedItem*40+95+15,420,
              MHighlightedItem*40+95+15);
         Line(220,MHighlightedItem*40+95+18,420,
              MHighlightedItem*40+95+18);
         Line(220,MHighlightedItem*40+95+21,420,
              MHighlightedItem*40+95+21);
         Line(220,MHighlightedItem*40+95+24,420,
              MHighlightedItem*40+95+24);
         Line(220,MHighlightedItem*40+95+27,420,
              MHighlightedItem*40+95+27);
      end;
end; {MHighlight}

procedure MMoveUp;
begin
   if MHighlightedItem > 1 then
      begin
         MHighlight(MHighlightedItem - 1);
         MSelectedItem := MSelectedItem - 1;
      end;
end; {MMoveUp}

procedure MMoveDown;
begin
   if (MHighlightedItem < MMaxSelections) and (MSelectedItem <
    MNumSelections) then
      begin
         MHighlight(MHighlightedItem + 1);
         MSelectedItem := MSelectedItem + 1;
      end;
end;
```

```
    procedure MGetInput;
    begin
      MCh := ReadKey;
      Case MCh of
          'h'              : Helpitem.GetHelp;
          chr(80),chr(50) : MMoveDown;
          chr(72),chr(56) : MMoveUp;
      end;
    end; {MGetInput}

begin
    MHighlight(1);
    repeat
      MGetInput
    until (MCh = #13) or (MCh = #27);
    if MCh = #27 then
      MGetChoice := 'null'
    else
      MGetChoice := MMenuSelection[MSelectedItem];
      MLastSelection := MSelectedItem;
end; {Mmenu.MgetChoice}

end.
```

```
unit NMenus;

interface

uses CRT, Graph, Help;

type
  Name = String[20];

  NMenu = object
    MenuTitle : String;
    X,Y,Color,TopItem,SelectedItem,HighLightedItem : Integer;
    NumSelections : Integer;
    MenuSelection : array[1..150] of Name;
    F : text;
    constructor Init(FileName : Name);
    function GetNumber : Integer;
  end;

var
  LastSelection : Integer;
  HelpItem      : Helps;

implementation

const
  MaxSelections = 8;

var
  Ch : Char;
  Counter : Integer;
  Liner   : Integer;

constructor NMenu.Init(FileName : Name);
begin
  HighlightedItem := 1;
  SelectedItem := 1;
  TopItem := 1;
  Counter := 0;
  Assign(F,FileName);
  Reset(F);
  Readln(F,MenuTitle);
  Readln(F,X);
```

```
    Readln(F,Y);
    Readln(F,Color);
    while (not eof(F)) and (Counter < 150) do
      begin
        Counter := Counter + 1;
        Readln(F,MenuSelection[Counter]);
      end;
    NumSelections := Counter;
    Close(F);
end;

function NMenu.GetNumber : Integer;

    procedure ShowMenu(Selection : Integer);
    begin
      SetColor(0);
      SetLineStyle(SolidLn,0,ThickWidth);
      SetWriteMode(CopyPut);
      SetTextJustify(CenterText,CenterText);
      for Liner := 1 to 18 do
          begin
            line(391,227+Liner,560,227+Liner);
          end;
      SetColor(Color);
      OutTextXY(X,Y,MenuTitle);
      Counter := 0;
      while (Counter + Selection <= NumSelections) and
            (Counter + Selection < Selection + MaxSelections) do
        begin
          OutTextXY(X,Counter*15+Y+22,
          MenuSelection[Counter + Selection]);
          Counter := Counter + 1;
        end;
      SetColor(0);
      SetLineStyle(SolidLn,0,ThickWidth);
      for Liner := 1 to 18 do
          begin
            line(X-75,Y+140+Liner,X+75,Y+140+Liner);
          end;
      SetColor(Color);
      if (Selection > 1) and (Selection <= NumSelections - 8) then
        OutTextXY(X,Y+150,'PgUp/PgDn')
      else if Selection > 1 then
```

```
      OutTextXY(X,Y+150,'PgUp')
   else if Selection <= NumSelections - 8 then
      OutTextXY(X,Y+150,'PgDn');
end; {ShowMenu}

procedure KillMenu;
begin
   if Color <> 0 then
      SetColor(0);
   SetWriteMode(CopyPut);
   Line(X-50,Y-3,X+50,Y-3);
   Line(X-50,Y,X+50,Y);
   Line(X-50,Y+3,X+50,Y+3);
   for Counter := 1 to 8 do
       begin
          Line(X-85,Counter*15+Y+3,X+85,Counter*15+Y+3);
          Line(X-85,Counter*15+Y+6,X+85,Counter*15+Y+6);
          Line(X-85,Counter*15+Y+9,X+85,Counter*15+Y+9);
       end;
   Line(X-35,Y+97,X+35,Y+97);
   Line(X-35,Y+100,X+35,Y+100);
   Line(X-35,Y+103,X+35,Y+103);
end; {KillMenu}

procedure Highlight(NewItem : Integer);
begin
   SetWriteMode(XorPut);
   SetColor(Color);
   Line(X-85,HighlightedItem*15+Y+3,X+85,HighlightedItem*15+Y+3);
   Line(X-85,HighlightedItem*15+Y+6,X+85,HighlightedItem*15+Y+6);
   Line(X-85,HighlightedItem*15+Y+9,X+85,HighlightedItem*15+Y+9);
   if HighLightedItem <> NewItem then
      begin
         HighlightedItem := NewItem;
         Line(X-85,NewItem*15+Y+3,X+85,NewItem*15+Y+3);
         Line(X-85,NewItem*15+Y+6,X+85,NewItem*15+Y+6);
         Line(X-85,NewItem*15+Y+9,X+85,NewItem*15+Y+9);
      end;
end; {Highlight}

procedure PageUp;
begin
   if TopItem > MaxSelections then
```

```
      begin
        KillMenu;
        SelectedItem := (SelectedItem - 8) - (HighlightedItem - 1);
        TopItem := TopItem - 8;
        HighlightedItem := 1;
        ShowMenu(TopItem);
        Highlight(HighlightedItem);
      end;
end; {PageUp}

procedure PageDown;
begin
  if TopItem + MaxSelections <= NumSelections then
      begin
        KillMenu;
        SelectedItem := (SelectedItem + 8) - (HighlightedItem - 1);
        TopItem := TopItem + 8;
        HighlightedItem := 1;
        ShowMenu(TopItem);
        Highlight(HighlightedItem);
      end;
end; {PageDown}

procedure MoveUp;
begin
  if HighlightedItem > 1 then
      begin
        Highlight(HighlightedItem - 1);
        SelectedItem := SelectedItem - 1;
      end;
end; {MoveUp}

procedure MoveDown;
begin
  if (HighlightedItem < MaxSelections)
      and (SelectedItem < NumSelections) then
      begin
        Highlight(HighlightedItem + 1);
        SelectedItem := SelectedItem + 1;
      end;
end; {MoveDown}

procedure GetInput;
```

```
    begin
      Ch := ReadKey;
      Case Ch of
          'h'              : HelpItem.GetHelp;
          chr(80),chr(50) : MoveDown;
          chr(72),chr(56) : MoveUp;
          chr(81),chr(51) : PageDown;
          chr(73),chr(57) : PageUp;
      end;
    end; {GetInput}

begin
    ShowMenu(1);
    Highlight(1);
    repeat
      GetInput
    until (Ch = #13) or (Ch = #27);
    if Ch = #27 then
      GetNumber := 0
    else
      GetNumber := SelectedItem;
    SetColor(0);
    SetLineStyle(SolidLn,0,ThickWidth);
    SetWriteMode(CopyPut);
    for Liner := 1 to 18 do
        begin
          Line(391,227+Liner,560,227+Liner);
        end;
    for Liner := 1 to 18 do
        begin
          Line(X-75,Y+140+Liner,X+75,Y+140+Liner);
        end;
     LastSelection := SelectedItem;
     KillMenu;
     TopItem := 1;
     SelectedItem := 1;
     HighlightedItem := 1;
end; {NMenu.GetNumber}

end.
```

```
unit Game;

interface

   procedure PlayGame;

implementation

uses CRT, Graph, Students, Helicopt, HpDialog, Menus, Help;

type
   Name = String[20];

var
   Counter1, Counter2, Counter3, MaxTime, MaxNum,
   PL1Score, PL2Score : Integer;
   Score : String;
   Ch, Pl : Char;
   LeftHPT, RightHPT : Helicopter;
   DialogScreen : array [1..3] of Dialog;
   WMenu : Menu;
   HelpItem : Helps;
   Choice : Name;

procedure ShowScores;
begin
   SetColor(0);
   SetWriteMode(CopyPut);
   Line(100,160,150,160);
   Line(100,162,150,162);
   Line(100,164,150,164);
   Line(100,166,150,166);
   Line(100,168,150,168);
   Line(100,170,150,170);
   Line(100,172,150,172);
   Line(100,174,150,174);
   Line(100,176,150,176);
   Line(100,178,150,178);
   Line(100,180,150,180);
   SetColor(12);
   Str(PL1Score,Score);
   OutTextXY(133,164,Score);
   SetColor(0);
```

```
    SetWriteMode(CopyPut);
    Line(210,160,280,160);
    Line(210,162,280,162);
    Line(210,164,280,164);
    Line(210,166,280,166);
    Line(210,168,280,168);
    Line(210,170,280,170);
    Line(210,172,280,172);
    Line(210,174,280,174);
    Line(210,176,280,176);
    Line(210,178,280,178);
    Line(210,180,280,180);
    SetColor(12);
    Str(PL2Score,Score);
    OutTextXY(243,164,Score);
    SetColor(0);
end;

procedure PlayOne;
begin
    DialogScreen[2].Init('NoGame.msg');
    DialogScreen[2].Show(350,50);
    SetColor(12);
    SetTextJustify(CenterText,CenterText);
    OutTextXY(440,180,LeftHPT.HPTInfo.HelicopterName);
    SetColor(0);
    Ch := ReadKey;
    while Ch = 'h' do
        begin
            HelpItem.GetHelp;
            Ch := ReadKey;
        end;
    DialogScreen[2].Hide;
    DialogScreen[2].Kill;
    LeftHPT.Hide;
    RightHPT.Hide;
    LeftHPT.Kill;
    RightHPT.Kill;
    PL2Score := PL2Score + 1;
    ShowScores;
    DialogScreen[2].Init('Ready.msg');
    DialogScreen[2].Show(350,50);
    Ch := ReadKey;
```

```
  while Ch = 'h' do
    begin
      HelpItem.GetHelp;
      Ch := ReadKey;
    end;
  DialogScreen[2].Hide;
  DialogScreen[2].Kill;
end;

procedure PlayTwo;
begin
  DialogScreen[1].Init('GScore.msg');
  DialogScreen[1].Show(72,50);
  ShowScores;
  DialogScreen[2].Init('Ready.msg');
  DialogScreen[2].Show(350,50);
  Ch := ReadKey;
  while Ch = 'h' do
    begin
      HelpItem.GetHelp;
      Ch := ReadKey;
    end;
  DialogScreen[2].Hide;
  DialogScreen[2].Kill;
  for Counter1 := 1 to 25 do
     begin
       Counter3 := 0;
       Counter2 := StudentModel.GetEntry(MaxNum);
       LeftHPT.Init(StudentModel.HPTArray[Counter2]);
       RightHPT.Init(StudentModel.HPTArray[Counter2]);
       LeftHPT.Show(72,225,StudentModel.HPTArray[Counter2]);
       RightHPT.Show(344,225,StudentModel.HPTArray[Counter2]);
       if Ch = #27 then Counter1 := 26;
       while (UpCase(Ch) <> 'O') and (UpCase(Ch) <> 'T') and (Ch <> #8) do
         begin
           Counter3 := Counter3 + 1;
           if Counter3 > MaxTime then
             Ch := #8;
           Delay(5);
           if KeyPressed then
             Ch := ReadKey;
         end;
       if Ch = #27 then Counter1 := 26;
```

```
Pl := UpCase(Ch);
if Pl = #8 then
   begin
      if MaxTime <> 1000 then
         PlayOne
      else
         begin
            LeftHPT.Hide;
            RightHPT.Hide;
            LeftHPT.Kill;
            RightHPT.Kill;
            DialogScreen[2].Init('TimeOut.msg');
            DialogScreen[2].Show(350,50);
            Ch := ReadKey;
            DialogScreen[2].Hide;
            DialogScreen[2].Kill;
         end
   end
else if (Counter1 <26) then
   begin
      RightHPT.Hide;
      RightHPT.Kill;
      if Pl = 'O' then
         DialogScreen[2].Init('Play1.msg')
      else DialogScreen[2].Init('Play2.msg');
         DialogScreen[2].Show(350,50);
      WMenu.Init('WETFUR.mnu');
      Choice := WMenu.GetChoice;
      DialogScreen[2].Hide;
      LeftHPT.Hide;
      DialogScreen[2].Kill;
      LeftHPT.Kill;
      if Choice <> LeftHPT.HPTInfo.HelicopterName then
         begin
            Sound(100);
            Delay(200);
            if Pl = 'O' then Dec(PL1Score)
            else Dec(PL2Score);
            NoSound;
         end
      else
         begin
            if Pl = 'O' then Inc(PL1Score)
```

```
                else Inc(Pl2Score);
              end;
            ShowScores;
            if Counter1 < 25 then
              begin
                DialogScreen[2].Init('Ready.msg');
                DialogScreen[2].Show(350,50);
                Ch := ReadKey;
                while Ch = 'h' do
                  begin
                    HelpItem.GetHelp;
                    Ch := ReadKey;
                  end;
                DialogScreen[2].Hide;
                DialogScreen[2].Kill;
              end;
          end;
        GoToXY(1,1);
        Ch := #13;
      end;
  if PL1Score > PL2Score then
    DialogScreen[2].Init('Win1.msg')
  else if PL2Score > PL1Score then
    DialogScreen[2].Init('Win2.msg')
  else DialogScreen[2].Init('Tie.msg');
  DialogScreen[2].Show(350,50);
  Ch := ReadKey;
  DialogScreen[2].Hide;
  DialogScreen[2].Kill;
  DialogScreen[1].Hide;
  DialogScreen[1].Kill;
end;

procedure PlayGame;
begin
  StudentModel.Update(' ','Game','Game',0);
  MaxNum := 150;
  DialogScreen[1].Init('Game.msg');
  DialogScreen[1].Show(350,60);
  Ch := ReadKey;
  while (Ch <> '1') and (Ch <> '2') and (Ch <> #27) do
    begin
      if Ch = 'h' then
```

```
            HelpItem.GetHelp;
          Ch := ReadKey;
        end;
    DialogScreen[1].Hide;
    DialogScreen[1].Kill;
    PL1Score := 0;
    PL2Score := 0;
    if Ch = #27 then Exit
    else if Ch = '1' then
      begin
        Randomize;
        MaxTime := Random(250) + 250;
        PlayTwo;
      end
    else if Ch = '2' then
      begin
        MaxTime := 1000;
        PlayTwo;
      end;
end;

end.
```

```
unit Utility;

interface

   procedure SetUp;

implementation

uses DOS,CRT,Graph,GX_TP,PCX_TP,Students,MMenus,Menus,HpScreen,Helicopt,
     HpDialog, Help;

type
   Name = String[20];

var
   retcode : Integer;
   F : Text;
   TextString, Ds : String;
   Counter : Integer;
   SetUpMenu : Menu;
   SetUpMMenu : Mmenu;
   DialogScreen : Dialog;
   StudentMenu : Menu;
   HPTMenu : Menu;
   StudentReport : Screen;
   HelpItem : helps;
   Choice : Name;
   Ch : Char;

procedure GetStudents;
var
   DirInfo : SearchRec;
begin
   Assign(F,'Student.rec');
   ReWrite(F);
   Writeln(F,'STUDENTS');
   Writeln(F,470);
   Writeln(F,230);
   Writeln(F,13);
   FindFirst('*.mdl',AnyFile, DirInfo);
   while DosError = 0 do
      begin
         Writeln(F,Copy(DirInfo.Name,1,5));
```

```
        FindNext(DirInfo);
      end;
  Close(F);
end;

procedure SelectHPT;
var
  HPTArray : array [1..150] of Name;
  DirInfo : SearchRec;
  F1,F2 : Text;
  Counter1 : Integer;
  retcode  : Integer;
  HPTName : Name;
begin
  SetUpMMenu.Mshowmenu('Menu.pcx');
  StudentModel.Mode := 'SelectHPT';
  Counter := 1;
  FindFirst('*.nam',AnyFile, DirInfo);
  while DosError = 0 do
    begin
      Assign(F1,Concat(Copy(DirInfo.Name,1,5),'_#1.dat'));
      Reset(F1);
      Readln(F1,HPTName);
      HPTArray[Counter] := HPTName;
      Counter := Counter + 1;
      Close(F1);
      FindNext(DirInfo);
    end;
  Assign(F,'Helicopt.rec');
  ReWrite(F);
  Writeln(F,'Helicopter');
  Writeln(F,470);
  Writeln(F,242);
  Writeln(F,13);
  for Counter1 := 1 to Counter-1 do
     if HPTArray[Counter1] <> '' then
       Writeln(F,HPTArray[Counter1]);
  Close(F);
  Assign(F,'HIntermd.def');
  Assign(F1,'WETFUR.mnu');
  Rewrite(F);
  Rewrite(F1);
  Writeln(F,'Intermed default');
```

```
Writeln(F,'Teach');
Writeln(F,'Intermediate');
Writeln(F,0);
Writeln(F,1);
Writeln(F,0);
Writeln(F1,'HELICOPTER');
Writeln(F1,470);
Writeln(F1,242);
Writeln(F1,13);
DialogScreen.Init('SelHPT.msg');
DialogScreen.Show(72,230);
HPTMenu.Init('Helicopt.rec');
Choice := HPTMenu.GetChoice;
while Choice <> 'null' do
  begin
    if Copy(Choice,5,1) = ' ' then
      begin
        Writeln(F,Concat(Copy(Choice,1,4),'__#1'));
        Writeln(F,Concat(Copy(Choice,1,4),'__#2'));
        Writeln(F,Concat(Copy(Choice,1,4),'__#3'));
      end
    else
      begin
        Writeln(F,Concat(Copy(Choice,1,5),'_#1'));
        Writeln(F,Concat(Copy(Choice,1,5),'_#2'));
        Writeln(F,Concat(Copy(Choice,1,5),'_#3'));
      end;
    Writeln(F1,Choice);
    for Counter1 := 1 to Counter-1 do
       if HPTArray[Counter1] = Choice then
         HPTArray[Counter1] := ' ';
    Assign(F2,'Helicopt.rec');
    ReWrite(F2);
    Writeln(F2,'HELICOPTER');
    Writeln(F2,470);
    Writeln(F2,230);
    Writeln(F2,13);
    for Counter1 := 1 to Counter-1 do
       if HPTArray[Counter1] <> ' ' then
         Writeln(F2,HPTArray[Counter1]);
    Close(F2);
    HPTMenu.Init('Helicopt.rec');
    Choice := HPTMenu.GetChoice;
```

```
      end;
   DialogScreen.Hide;
   DialogScreen.Kill;
   Close(F);
   Close(F1);
end;


procedure AddHPT;
const
   ALPHA = ['0'..'9','A'..'Z','a'..'z','-'];
var
   HPTName : Name;
   DirInfo : SearchRec;
   Counter1,Counter2 : Integer;
   Size : Word;
   P : Pointer;
   S : String;
   FileName : String;
   F : File;
   F1 : Text;
   Ch : Char;
   Offset,MaxX,MaxY,X,Y : Integer;
   OldHPT : Helicopter;
   Liner : Integer;
begin
    SetUpMMenu.Mshowmenu('Menu.pcx');
    DialogScreen.Init('CheckHP.msg');
    DialogScreen.Show(205,60);
    Ch := ReadKey;
    if Ch = #27 then
    begin
       Exit
    end;
    DialogScreen.Hide;
    DialogScreen.Kill;
    StudentModel.Mode := 'AddHPT';
    HPTName := '';
    Counter1 := 1;
    SetLineStyle(SolidLn,0,ThickWidth);
    SetWriteMode(CopyPut);
    SetColor(13);
    DialogScreen.Init('GetHPT.msg');
```

```
DialogScreen.Show(205,60);
Line(220,182,420,182);
Line(220,184,420,184);
Line(220,186,420,186);
Line(220,188,420,188);
Line(220,190,420,190);
Line(220,192,420,192);
Ch := #8;
while ((Ch <> #13) and (Counter1 < 21)) or (HPTName = '') do
   begin
      SetColor(10);
      Ch := ReadKey;
      if Ch = chr(32) then
         begin
            HPTName := Concat(HPTName,chr(32));
            Counter1 := Counter1 + 1;
         end;
      if Ch in ALPHA then
         begin
            SetColor(10);
            OutTextXY(225+10*Counter1,186,Ch);
            HPTName := Concat(HPTName,Ch);
            Counter1 := Counter1 + 1;
            {SetColor(10);}
         end;
      if (Ch = #8) and (Counter1 > 1) then
        begin
            SetWriteMode(CopyPut);
            SetColor(13);
            Line(210+10*Counter1,182,220+10*Counter1,182);
            Line(210+10*Counter1,184,220+10*Counter1,184);
            Line(210+10*Counter1,186,220+10*Counter1,186);
            Line(210+10*Counter1,188,220+10*Counter1,188);
            Line(210+10*Counter1,190,220+10*Counter1,190);
            Line(210+10*Counter1,192,220+10*Counter1,192);
            Counter1 := Counter1 - 1;
            HPTName := Copy(HPTName,1,Counter1-1);
        end;
      if HPTName = '' then begin
        SetColor(12);
        SetTextJustify(CenterText,CenterText);
        OutTextXY(320,425,'File Name inputs ERROR, Try again !!');
        Ch := ReadKey;
```

```
            SetWriteMode(CopyPut);
            SetColor(0);
            for Liner := 0 to 20 do
            begin
               Line(100,415+Liner,540,415+Liner);
            end;
         end;
      SetColor(10);
      end;
DialogScreen.Hide;
DialogScreen.Kill;
DialogScreen.Init('New.msg');
DialogScreen.Show(72,50);
SetColor(LightRed);
SetTextJustify(CenterText,CenterText);
OutTextXY(185,125,HPTName);
if Copy(HPTName,5,1) = ' ' then
   FindFirst(Concat(Copy(HPTName,1,4),'_.nam'),AnyFile, DirInfo)
else FindFirst(Concat(Copy(HPTName,1,5),'.nam'),AnyFile, DirInfo);
Size := ImageSize(0,0,230,150);
GetMem(P,Size);
GetImage(72,50,302,200,P^);
Assign(F,Concat(Copy(HPTName,1,4),'_.nam'))
DialogScreen.Hide;
DialogScreen.Kill;
Offset := 140;
MaxX := 639;
MaxY := 219;
for Counter1 := 1 to 3 do
   begin
      X := 0;
      Y := 0;
      str(Counter1,s);
      if Copy(HPTName,5,1) = ' ' then
         retcode := pcxFileDisplay(Concat(Copy(HPTName,1,4),'__#',s,
                              '.pcx',72,230,0)
      else
         retcode := pcxFileDisplay(Concat(Copy(HPTName,1,5),'_#',s,
                              '.pcx'),72,230,0);
      if retcode <> gxSUCCESS then
      begin
         DialogScreen.Init('Nopcx.msg');
         DialogScreen.Show(205,60);
```

```
    Ch := ReadKey;
    DialogScreen.Hide;
    DialogScreen.Kill;
    FreeMem(P,Size);
    exit
  end
  else
  begin
    if Copy(HPTName,5,1) = ' ' then
      Assign(F,Concat(Copy(HPTName,1,4),'_.nam'))
    else Assign(F,Concat(Copy(HPTName,1,5),'.nam'));
    ReWrite(F,1);
    BlockWrite(F,P^,Size);
    Close(F);
    if Counter = 1 then
      FreeMem(P,Size);
    {Input the data for this view}
    if Copy(HPTName,5,1) = ' ' then
       Assign(F1,Concat(Copy(HPTName,1,4),'__#',s,'.dat'))
    else Assign(F1,Concat(Copy(HPTName,1,5),'_#',s,'.dat'));
    ReWrite(F1);
    Writeln(F1,HPTName);
    Writeln(F1,'');
    Writeln(F1,'');
    SetUpMenu.Init('Wing.mnu');
    Choice := SetUpMenu.GetChoice;
    if Choice <> 'null' then
      begin
        Writeln(F1,Choice);
        Str(LastSelection,s);
        if LastSelection < 10 then
          Writeln(F1,Concat('Wing0',s))
        else Writeln(F1,Concat('Wing1',s));
      end
    else
      begin
        Writeln(F1,'');
        Writeln(F1,'');
      end;

    SetUpMenu.Init('Engi.mnu');
    Choice := SetUpMenu.GetChoice;
    if Choice <> 'null' then
```

```
   begin
      Writeln(F1,Choice);
      Str(LastSelection,s);
      if LastSelection < 10 then
         Writeln(F1,Concat('Engi0',s))
      else Writeln(F1,Concat('Engi1',s));
   end
else
   begin
      Writeln(F1,'');
      Writeln(F1,'');
   end;

SetUpMenu.Init('Fuse.mnu');
Choice := SetUpMenu.GetChoice;
if Choice <> 'null' then
   begin
      Writeln(F1,Choice);
      Str(LastSelection,s);
      if LastSelection < 10 then
         Writeln(F1,Concat('Fuse0',s))
      else Writeln(F1,Concat('Fuse1',s));
   end
else
   begin
      Writeln(F1,'');
      Writeln(F1,'');
   end;

SetUpMenu.Init('Trot.mnu');
Choice := SetUpMenu.GetChoice;
if Choice <> 'null' then
    begin
       Writeln(F1,Choice);
       Str(LastSelection,s);
       if LastSelection < 10 then
          Writeln(F1,Concat('Trot0',s))
       else Writeln(F1,Concat('Trot1',s));
    end
else
    begin
       Writeln(F1,'');
       Writeln(F1,'');
```

```
    end;

SetColor(0);
SetLineStyle(SolidLn,0,ThickWidth);
for Liner := 1 to 18 do
begin
    line(385,242+Liner,560,242+Liner);
end;
SetUpMenu.Init('Mrot.mnu');
Choice := SetUpMenu.GetChoice;
if Choice <> 'null' then
    begin
        Writeln(F1,Choice);
        Str(LastSelection,s);
        if LastSelection < 10 then
            Writeln(F1,Concat('Mrot0',s))
        else Writeln(F1,Concat('Mrot1',s));
    end
else
    begin
        Writeln(F1,'');
        Writeln(F1,'');
    end;

SetColor(0);
SetLineStyle(SolidLn,0,ThickWidth);
for Liner := 1 to 18 do
begin
    line(385,242+Liner,560,242+Liner);
end;
SetUpMenu.Init('Ucag.mnu');
Choice := SetUpMenu.GetChoice;
if Choice <> 'null' then
    begin
        Writeln(F1,Choice);
        Str(LastSelection,s);
        if LastSelection < 10 then
            Writeln(F1,Concat('Ucag0',s))
        else Writeln(F1,Concat('Ucag1',s));
    end
else
    begin
        Writeln(F1,'');
```

```
        Writeln(F1,'');
    end;

SetColor(0);
SetLineStyle(SolidLn,0,ThickWidth);
for Liner := 1 to 18 do
begin
    line(385,242+Liner,560,242+Liner);
end;
SetUpMenu.Init('Hstl.mnu');
Choice := SetUpMenu.GetChoice;
if Choice <> 'null' then
    begin
        Writeln(F1,Choice);
        Str(LastSelection,s);
        if LastSelection < 10 then
            Writeln(F1,Concat('Hstl0',s))
        else Writeln(F1,Concat('Hstl1',s));
    end
else
    begin
        Writeln(F1,'');
        Writeln(F1,'');
    end;

SetColor(0);
SetLineStyle(SolidLn,0,ThickWidth);
for Liner := 1 to 18 do
begin
    line(385,242+Liner,560,242+Liner);
end;
SetUpMenu.Init('Hstn.mnu');
Choice := SetUpMenu.GetChoice;
if Choice <> 'null' then
    begin
        Writeln(F1,Choice);
        Str(LastSelection,s);
        if LastSelection < 10 then
            Writeln(F1,Concat('Hstn0',s))
        else Writeln(F1,Concat('Hstn1',s));
    end
else
    begin
```

```
                    Writeln(F1,'');
                    Writeln(F1,'');
                end;
            SetColor(0);
            SetLineStyle(SolidLn,0,ThickWidth);
            for Liner := 1 to 18 do
            begin
                line(385,242+Liner,560,242+Liner);
            end;
            Close(F1);
        end;

    end;
end;

procedure Report;
begin
    SetUpMMenu.Mshowmenu('Menu.pcx');
    StudentModel.Mode := 'StuRep';
    GetStudents;
    DialogScreen.Init('StuRep.msg');
    DialogScreen.Show(72,230);
    StudentMenu.Init('Student.rec');
    Choice := StudentMenu.GetChoice;
    while Choice <> 'null' do
        begin
            SetUpMMenu.Mshowmenu('Report.pcx');
            Assign(F,Concat(Choice,'.mdl'));
            Reset(F);
            SetColor(12);
            SetTextJustify(LeftText,LeftText);
            OutTextXY(340,150,Choice);
            for Counter := 1 to 4 do
                begin
                    Readln(F,TextString);
                    if TextString = '0' then OutTextXY(340,Counter*32+140,'Not
Tested')
                    else OutTextXY(340,Counter*32+140,TextString);
                end;
            SetColor(0);
            Close(F);
            Ch := ReadKey;
            while Ch = 'h' do
```

```
                begin
                    HelpItem.GetHelp;
                    Ch := ReadKey;
                end;
            DialogScreen.Hide;
            DialogScreen.Kill;
            ClearDevice;
            SetUpMMenu.Mshowmenu('Menu.pcx');
            DialogScreen.Init('StuRep.msg');
            DialogScreen.Show(72,230);
            GetStudents;
            StudentMenu.Init('Student.rec');
            Choice := StudentMenu.GetChoice;
        end;
    DialogScreen.Hide;
    DialogScreen.Kill;
end;

procedure SetUp;
const
   ALPHA = ['0'..'9','A'..'Z','a'..'z','-'];
var
   PassWord : Name;
   Counter1 : Integer;
begin
   ClearDevice;
   PassWord := '';
   Counter1 := 1;
   SetLineStyle(SolidLn,0,ThickWidth);
   SetWriteMode(CopyPut);
   SetColor(13);
   DialogScreen.Init('Passwd.msg');
   DialogScreen.Show(205,60);
   Line(220,182,420,182);
   Line(220,184,420,184);
   Line(220,186,420,186);
   Line(220,188,420,188);
   Line(220,190,420,190);
   Line(220,192,420,192);
   Ch := #8;
   while (Ch <> #13) and (Counter1 < 21) do
      begin
         Ch := ReadKey;
```

```
      if Ch = chr(32) then
        begin
          PassWord := Concat(PassWord,chr(32));
          Counter1 := Counter1 + 1;
        end;
      if Ch in ALPHA then
        begin
          SetColor(10);
          OutTextXY(225+10*Counter1,186,Ch);
          PassWord := Concat(PassWord,Ch);
          Counter1 := Counter1 + 1;
        end;
      if (Ch = #8) and (Counter1 > 1) then
        begin
          SetWriteMode(CopyPut);
          SetColor(13);
          Line(210+10*Counter1,182,220+10*Counter1,182);
          Line(210+10*Counter1,184,220+10*Counter1,184);
          Line(210+10*Counter1,186,220+10*Counter1,186);
          Line(210+10*Counter1,188,220+10*Counter1,188);
          Line(210+10*Counter1,190,220+10*Counter1,190);
          Line(210+10*Counter1,192,220+10*Counter1,192);
          Counter1 := Counter1 - 1;
          PassWord := Copy(PassWord,1,Counter1-1);
          SetColor(13);
        end;
    end;
DialogScreen.Hide;
DialogScreen.Kill;
if PassWord <> '500313' then
  Exit;
StudentModel.Mode := 'Setup';
SetUpMMenu.MInit('SetUp1.mnu');
SetUpMMenu.Mshowmenu('SetUp1.pcx');
Choice := SetUpMMenu.MGetChoice;
while (Choice <> 'EXIT') and (Choice <> 'null') do
  begin
    if Choice = 'SELECT HELICOPTER' then
      begin
        ClearDevice;
        SelectHPT;
      end;
    if Choice = 'ADD/MODIF HELICOPTER' then
```

```
            begin
               ClearDevice;
               DialogScreen.Init('Hpmsg.msg');
               DialogScreen.Show(205,60);
               Ch := ReadKey;
               DialogScreen.Hide;
               DialogScreen.Kill;
               AddHPT;
            end;
         if Choice = 'STUDENT REPORT' then
            begin
               ClearDevice;
               Report;
            end;
         GotoXY(1,1);
         StudentModel.Mode := 'Setup';
         SetUpMMenu.MInit('SetUp1.mnu');
         SetUpMMenu.Mshowmenu('SetUp1.pcx');
         Choice := SetUpMMenu.MGetChoice;
      end;
end;

end.
```

```
unit HpTutor;

interface

   procedure Tutor;

implementation

uses CRT, Graph, Students, Helicopt, HpDialog, Menus, Help, MMenus, NMenus;

type
   Name = String[20];

var
   Counter1, Counter2, Counter3, MaxNum, ChoiceNum, Score : Integer;
   Comparison : Real;
   Ch : Char;
   LeftHPT, RightHPT : Helicopter;
   DialogScreen : array [1..5] of Dialog;
   WMenu : Menu;
   WNMenu : NMenu;
   Mainmenu : Mmenu;
   HelpItem : Helps;
   StuName, Choice, FourthCh : Name;
   CorrectAnswer, CloseAnswer, Done, Dialog_Kill : Boolean;
   S : String[20];
   Liner : Integer;
   Num_s : String[3];
   Tptr1,Tptr2,Tptr3,Tptr4,Tptr5 : Pointer;
   C : Integer;

procedure ShowFeature(Feature : Name);
begin
   if Feature <> '' then
      begin
         DialogScreen[4].Init(Concat(Feature,'.msg'));
         DialogScreen[4].Show(350,230);
         Ch := ReadKey;
         Dialog_Kill := False;
         while (Ch = 'h') do
            begin
               if StudentModel.Level = 'Novice' then
                  begin
```

```
                DialogScreen[1].Hide;
                DialogScreen[1].Kill;
                DialogScreen[4].Hide;
                DialogScreen[4].Kill;
                Dialog_Kill := True;
                HelpItem.GetHelp;
                Ch := #8;
              end;
            if StudentModel.Level = 'Intermediate' then
              begin
                DialogScreen[1].Hide;
                DialogScreen[1].Kill;
                DialogScreen[4].Hide;
                DialogScreen[4].Kill;
                Dialog_Kill := True;
                DialogScreen[2].Hide;
                DialogScreen[2].Kill;
                HelpItem.GetHelp;
                Ch := #8;
                DialogScreen[2].Init(Concat(Copy
                              (StudentModel.HPTArray[Counter1],1,5),'.nam'));
                DialogScreen[2].Show(72,50);
              end;
            Ch := #8;
          end;
          if not Dialog_Kill then
            begin
              DialogScreen[4].Hide;
              DialogScreen[4].Kill;
            end;
      end;
end;

function CompareHelicopter : Real;
begin
  Comparison := 0;
  Counter3 := 0;
  for Counter2 := 1 to 1 do
     if (LeftHPT.HPTInfo.Wings[Counter2] <> '') and
       (RightHPT.HPTInfo.Wings[Counter2] <> '') then
       if LeftHPT.HPTInfo.Wings[Counter2] = RightHPT.HPTInfo.Wings[Counter2]
then
          begin
```

```
            Comparison := Comparison + 1;
            Inc(Counter3);
          end
        else Inc(Counter3);
for Counter2 := 1 to 1 do
    if (LeftHPT.HPTInfo.Engine[Counter2] <> '') and
      (RightHPT.HPTInfo.Engine[Counter2] <> '') then
       if LeftHPT.HPTInfo.Engine[Counter2] =
                              RightHPT.HPTInfo.Engine[Counter2] then
          begin
            Comparison := Comparison + 1;
            Inc(Counter3);
          end
       else Inc(Counter3);
for Counter2 := 1 to 1 do
    if (LeftHPT.HPTInfo.Fuslag[Counter2] <> '') and
      (RightHPT.HPTInfo.Fuslag[Counter2] <> '') then
      if LeftHPT.HPTInfo.Fuslag[Counter2] = RightHPT.HPTInfo.Fuslag[Counter2]
        then
          begin
            Comparison := Comparison + 1;
            Inc(Counter3);
          end
        else Inc(Counter3);
for Counter2 := 1 to 1 do
    if (LeftHPT.HPTInfo.Trot[Counter2] <> '') and
      (RightHPT.HPTInfo.Trot[Counter2] <> '') then
      if LeftHPT.HPTInfo.Trot[Counter2] = RightHPT.HPTInfo.Trot[Counter2] then
          begin
            Comparison := Comparison + 1;
            Inc(Counter3);
          end
        else Inc(Counter3);
for Counter2 := 1 to 1 do
    if (LeftHPT.HPTInfo.Mrot[Counter2] <> '') and
      (RightHPT.HPTInfo.Mrot[Counter2] <> '') then
      if LeftHPT.HPTInfo.Mrot[Counter2] = RightHPT.HPTInfo.Mrot[Counter2]
     then
          begin
            Comparison := Comparison + 1;
            Inc(Counter3);
          end
        else Inc(Counter3);
```

```
  for Counter2 := 1 to 1 do
    if (LeftHPT.HPTInfo.Ucag[Counter2] <> '') and
      (RightHPT.HPTInfo.Ucag[Counter2] <> '') then
      if LeftHPT.HPTInfo.Ucag[Counter2] = RightHPT.HPTInfo.Ucag[Counter2]
        then
        begin
          Comparison := Comparison + 1;
          Inc(Counter3);
        end
      else Inc(Counter3);
  for Counter2 := 1 to 1 do
    if (LeftHPT.HPTInfo.Hstl[Counter2] <> '') and
      (RightHPT.HPTInfo.Hstl[Counter2] <> '') then
      if LeftHPT.HPTInfo.Hstl[Counter2] = RightHPT.HPTInfo.Hstl[Counter2] then
        begin
          Comparison := Comparison + 1;
          Inc(Counter3);
        end
      else Inc(Counter3);
  for Counter2 := 1 to 1 do
    if (LeftHPT.HPTInfo.Hstn[Counter2] <> '') and
      (RightHPT.HPTInfo.Hstn[Counter2] <> '') then
      if LeftHPT.HPTInfo.Hstn[Counter2] = RightHPT.HPTInfo.Hstn[Counter2] then
        begin
          Comparison := Comparison + 1;
          Inc(Counter3);
        end
      else Inc(Counter3);
  if Counter3 <> 0 then
    CompareHelicopter := Comparison/Counter3;
end;

procedure Diagnose;
const
  ALPHA = ['A'..'Z','a'..'z'];
begin
  StudentModel.Mode := 'Diagnose';
  StuName := '';
  Counter1 := 1;
  SetLineStyle(SolidIn,0,ThickWidth);
  SetWriteMode(CopyPut);
  SetColor(13);
  DialogScreen[1].Init('GetName.msg');
```

```
DialogScreen[1].Show(205,60);
SetLineStyle(Solidln,0,ThickWidth);
Line(220,162,420,162);
Line(220,165,420,165);
Line(220,168,420,168);
Line(220,171,420,171);
Line(220,174,420,174);
while (Ch <> #13) and (Counter1 < 21) do
   begin
      SetColor(10);
      Ch := ReadKey;
      if Ch = chr(32) then
         begin
            StuName := Concat(StuName,chr(32));
            Counter1 := Counter1 + 1;
         end;
      if Ch in ALPHA then
         begin
            SetColor(10);
            OutTextXY(220+10*Counter1,166,Ch);
            StuName := Concat(StuName,Ch);
            Counter1 := Counter1 + 1;
            SetColor(13);
         end;
      if (Ch = #8) and (Counter1 > 1) then
         begin
            SetWriteMode(CopyPut);
            SetColor(13);
            Line(210+10*Counter1,162,220+10*Counter1,162);
            Line(210+10*Counter1,165,220+10*Counter1,165);
            Line(210+10*Counter1,168,220+10*Counter1,168);
            Line(210+10*Counter1,171,220+10*Counter1,171);
            Line(210+10*Counter1,174,220+10*Counter1,174);
            Counter1 := Counter1 - 1;
            StuName := Copy(StuName,1,Counter1-1);
            SetColor(13);
         end;
   end;
DialogScreen[1].Hide;
DialogScreen[1].Kill;
SetColor(0);
StudentModel.Update(StuName,'Teach','Novice',0);
StudentModel.Mode := 'Diagnose';
```

```
MaxNum := 75;
Counter3 := 0;
DialogScreen[1].Init('Welcome.msg');
DialogScreen[1].Show(205,60);
Ch := ReadKey;
while Ch = 'h' do
   begin
      HelpItem.GetHelp;
      Ch := ReadKey;
   end;
DialogScreen[1].Hide;
DialogScreen[1].Kill;
if Ch = #27 then Exit;
DialogScreen[1].Init('Diagnose.msg');
DialogScreen[1].Show(205,60);
Ch := ReadKey;
while Ch = 'h' do
   begin
      HelpItem.GetHelp;
      Ch := ReadKey;
   end;
DialogScreen[1].Hide;
DialogScreen[1].Kill;
if Ch = #27 then Exit;
for Counter2 := 1 to 10 do
    begin
       Counter1 := StudentModel.GetEntry(MaxNum);
       if StudentModel.HPTArray[Counter1] <> '' then
          begin
             LeftHPT.Init(StudentModel.HPTArray[Counter1]);
             if LeftHPT.HPTInfo.ExampleInfo <> '' then
                begin
                   LeftHPT.Show(72,225,StudentModel.HPTArray[Counter1]);
                  WNMenu.Init(Concat(Copy(LeftHPT.HPTInfo.ExampleInfo,
                            1,4),'.mnu'));
                   Str(WNmenu.GetNumber,Num_s);
                   Num_s := Concat('0',Num_s);
                   Choice := (Concat(Copy(LeftHPT.HPTInfo.ExampleInfo,1,4),
                         Num_s));
                   SetColor(0);
                   SetLineStyle(SolidLn,0,ThickWidth);
                   for Liner := 1 to 18 do
                      begin
```

```
                    Line(391,242+Liner,560,242+Liner);
                  end;
                if Choice = 'null' then
                  begin
                    LeftHPT.Hide;
                    LeftHPT.Kill;
                    Counter3 := Counter3 + 1;
                    Exit;
                  end;
                if Choice <> LeftHPT.HPTInfo.ExampleInfo then
                  begin
                    Counter3 := Counter3 + 1;
                    Sound(100);
                    Delay(200);
                    NoSound;
                  end;
                LeftHPT.Hide;
              end;
              LeftHPT.Kill;
            end;
          GoToXY(1,1);
        end;
    if Counter3 <= 1 then
      begin
        StudentModel.Update(StuName,'Teach','Intermediate',0);
      end
    else
      begin
        StudentModel.Update(StuName,'Teach','Novice',0);
      end;
end;


procedure Teach;
begin
  Ch := #13;
  if StudentModel.Level = 'Novice' then
    MaxNum := 75
  else MaxNum := 150;
  if StudentModel.Level = 'Novice' then
    DialogScreen[1].Init('TeaNov.msg')
  else DialogScreen[1].Init('TeaInt.msg');
    DialogScreen[1].Show(205,60);
```

```
Ch := ReadKey;
while Ch = 'h' do
  begin
    HelpItem.GetHelp;
    Ch := ReadKey;
  end;
DialogScreen[1].Hide;
DialogScreen[1].Kill;
if Ch = #27 then Exit;
for Counter1 := StudentModel.NumShown to MaxNum do
   if StudentModel.Level = 'Novice' then
     begin
       if StudentModel.HPTArray[Counter1] <> '' then
         begin
           LeftHPT.Init(StudentModel.HPTArray[Counter1]);
           if LeftHPT.HPTInfo.ExampleInfo <> '' then
             begin
               LeftHPT.Show(72,225,StudentModel.HPTArray[Counter1]);
               ShowFeature(LeftHPT.HPTInfo.ExampleInfo);
               LeftHPT.Hide;
             end;
           LeftHPT.Kill;
           if Ch = #27 then Exit;
             Inc(StudentModel.NumShown);
         end;
     end
   else
     begin
       if StudentModel.HPTArray[Counter1] <> '' then
         begin
           LeftHPT.Init(StudentModel.HPTArray[Counter1]);
           LeftHPT.Show(72,225,StudentModel.HPTArray[Counter1]);
                 DialogScreen[2].Init(Concat(Copy
                       (StudentModel.HPTArray[Counter1],1,5),'.nam'));
           DialogScreen[2].Show(70,50);
           for Counter2 := 1 to 1 do
              if Ch <> #27 then
                ShowFeature(LeftHPT.HPTInfo.WingsInfo[Counter2]);
           for Counter2 := 1 to 1 do
              if Ch <> #27 then
                ShowFeature(LeftHPT.HPTInfo.EngineInfo[Counter2]);
           for Counter2 := 1 to 1 do
              if Ch <> #27 then
```

```
                    ShowFeature(LeftHPT.HPTInfo.FuslagInfo[Counter2]);
                for Counter2 := 1 to 1 do
                    if Ch <> #27 then
                    ShowFeature(LeftHPT.HPTInfo.TrotInfo[Counter2]);
                for Counter2 := 1 to 1 do
                    if Ch <> #27 then
                    ShowFeature(LeftHPT.HPTInfo.MrotInfo[Counter2]);
                for Counter2 := 1 to 1 do
                    if Ch <> #27 then
                    ShowFeature(LeftHPT.HPTInfo.UcagInfo[Counter2]);
                for Counter2 := 1 to 1 do
                    if Ch <> #27 then
                    ShowFeature(LeftHPT.HPTInfo.HstnInfo[Counter2]);
                for Counter2 := 1 to 1 do
                    if Ch <> #27 then
                    ShowFeature(LeftHPT.HPTInfo.HstlInfo[Counter2]);
                DialogScreen[2].Hide;
                LeftHPT.Hide;
                DialogScreen[2].Kill;
                LeftHPT.Kill;
                if Ch = #27 then Exit;
                  Inc(StudentModel.NumShown);
            end;
        end;
    StudentModel.NumShown := MaxNum;
end;

procedure ReviewNovice;
begin
  MaxNum := 75;
  Done := False;
  DialogScreen[1].Init('Return.msg');
  DialogScreen[1].Show(205,60);
  Ch := ReadKey;
  while Ch = 'h' do
    begin
      HelpItem.GetHelp;
      Ch := ReadKey;
    end;
  DialogScreen[1].Hide;
  DialogScreen[1].Kill;
  if Ch = #27 then Exit;
  DialogScreen[1].Init('RevNov.msg');
```

```
DialogScreen[1].Show(205,60);
Ch := ReadKey;
while Ch = 'h' do
  begin
    HelpItem.GetHelp;
    Ch := ReadKey;
  end;
DialogScreen[1].Hide;
DialogScreen[1].Kill;
if Ch = #27 then Exit;
Counter1 := StudentModel.GetEntry(MaxNum);
while (Counter1 <> 0) and (Done = False) do
  begin
    LeftHPT.Init(StudentModel.HPTArray[Counter1]);
    if LeftHPT.HPTInfo.ExampleInfo <> '' then
      begin
        LeftHPT.Show(72,225,StudentModel.HPTArray[Counter1]);
        WNMenu.Init(Concat(Copy(LeftHPT.HPTInfo.ExampleInfo,1,4),'.mnu'));
        Str(WNmenu.GetNumber,Num_s);
        Num_s := Concat('0',Num_s);
        Choice := (Concat(Copy(LeftHPT.HPTInfo.ExampleInfo,1,4),Num_s));
        SetColor(0);
        SetLineStyle(SolidLn,0,ThickWidth);
        for Liner := 1 to 18 do
          begin
            Line(391,242+Liner,560,242+Liner);
          end;
        if Choice = LeftHPT.HPTInfo.ExampleInfo then
          begin
            StudentModel.HPTArray[Counter1] := '';
            DialogScreen[1].Init(Concat('Correct',Chr(Random(10)+48),'.msg'));
            DialogScreen[1].Show(350,50);
            Ch := ReadKey;
            while Ch = 'h' do
              begin
                HelpItem.GetHelp;
                Ch := ReadKey;
              end;
            DialogScreen[1].Hide;
            DialogScreen[1].Kill;
          end
        else if Choice <> 'null' then
          begin
```

```
                if StudentModel.AddEntry(StudentModel.HPTArray[Counter1],
                        MaxNum) = False then
                  Done := True;
                DialogScreen[1].Init(Concat('Wrong',Chr(Random(10)+48),'.msg'));
                DialogScreen[1].Show(350,50);
                ShowFeature(LeftHPT.HPTInfo.ExampleInfo);
                if not Dialog_Kill then
                  begin
                    DialogScreen[1].Hide;
                    DialogScreen[1].Kill;
                  end;
                Ch := ReadKey;
              end;
               LeftHPT.Hide;
            end
          else StudentModel.HPTArray[Counter1] := '';
          LeftHPT.Kill;
          if (Ch = #27) or (Choice = 'null') then Exit;
          Counter1 := StudentModel.GetEntry(MaxNum);
      end;
  Done := True;
end;

procedure ReviewIntermediate;

procedure HandleCorrectIntermediate;
begin
  SetColor(14);
  Dialog_Kill := False;
  DialogScreen[1].Init(Concat('Correct',Chr(Random(10)+48),'.msg'));
  DialogScreen[1].Show(350,50);
  DialogScreen[2].Init(Concat(Copy(StudentModel.HPTArray[Counter1],1,5),'.nam'));
  DialogScreen[2].Show(72,50);
  StudentModel.HPTArray[Counter1] := '';
  Ch := ReadKey;
  while Ch = 'h' do
    begin
      DialogScreen[2].Hide;
      DialogScreen[2].Kill;
      DialogScreen[1].Hide;
      DialogScreen[1].Kill;
      Dialog_Kill := True;
      HelpItem.GetHelp;
```

```
        Ch := ReadKey;
      end;
   if not Dialog_Kill then
      begin
        DialogScreen[2].Hide;
        DialogScreen[2].Kill;
        DialogScreen[1].Hide;
        DialogScreen[1].Kill;
       end;
end;

procedure HandleWrongIntermediate;
begin
   DialogScreen[1].Init(Concat('Wrong',Chr(Random(10)+48),'.msg'));
   DialogScreen[1].Show(350,50);
   Ch := Readkey;
   if Ch = 'h' then
      HelpItem.GetHelp;
   DialogScreen[1].Hide;
   DialogScreen[1].Kill;
   if Ch = #27 then Exit;
   DialogScreen[1].Init('IdWEFT.msg');
   DialogScreen[1].Show(350,50);
   DialogScreen[2].Init(Concat(Copy(StudentModel.HPTArray[Counter1],1,5),'.nam'));
   DialogScreen[2].Show(72,50);
   Ch := ReadKey;
   while Ch = 'h' do
      begin
        DialogScreen[1].Hide;
        DialogScreen[1].Kill;
        Dialog_Kill := True;
        DialogScreen[2].Hide;
        DialogScreen[2].Kill;
        HelpItem.GetHelp;
        Ch := ' ';
        DialogScreen[2].Init(Concat(Copy(StudentModel.HPTArray[Counter1],1,5),
                          '.nam'));
        DialogScreen[2].Show(72,50);
      end;
   if not Dialog_Kill then
      begin
        DialogScreen[1].Hide;
        DialogScreen[1].Kill;
```

```
  end;
for Counter2 := 1 to 1 do
   if (LeftHPT.HPTInfo.Wings[Counter2] <> '') and (Ch <> #27) then
      begin
         WMenu.Init(Concat(Copy(LeftHPT.HPTInfo.WingsInfo[Counter2],1,4),
               '.mnu'));
         Choice := WMenu.GetChoice;
         if Choice = LeftHPT.HPTInfo.Wings[Counter2] then
            begin
               Dialog_Kill := False;
               DialogScreen[1].Init(Concat('Correct',Chr(Random(10)+48),'.msg'));
               DialogScreen[1].Show(350,50);
               Ch := ReadKey;
               while Ch = 'h' do
                  begin
                     DialogScreen[1].Hide;
                     DialogScreen[1].Kill;
                     Dialog_Kill := True;
                     DialogScreen[2].Hide;
                     DialogScreen[2].Kill;
                     HelpItem.GetHelp;
                     Ch := ' ';
                     DialogScreen[2].Init(Concat(Copy(
                           StudentModel.HPTArray[Counter1],1,5),'.nam'));
                     DialogScreen[2].Show(72,50);
                  end;
               if not Dialog_Kill then
                  begin
                     DialogScreen[1].Hide;
                     DialogScreen[1].Kill;
                  end;
            end
            else if Choice <> 'null' then
               begin
                  Dialog_Kill := False;
                  DialogScreen[1].Init(Concat('Wrong',Chr(Random(10)+48),'.msg'));
                  DialogScreen[1].Show(350,50);
                  ShowFeature(LeftHPT.HPTInfo.WingsInfo[Counter2]);
                  if not Dialog_Kill then
                     begin
                        DialogScreen[1].Hide;
                        DialogScreen[1].Kill;
                     end;
```

```
            end
          else Ch := #27;
      end;
for Counter2 := 1 to 1 do
   if (LeftHPT.HPTInfo.Engine[Counter2] <> '') and (Ch <> #27) then
      begin
        WMenu.Init(Concat(Copy(LeftHPT.HPTInfo.EngineInfo[Counter2],1,4),
              '.mnu'));
        Choice := WMenu.GetChoice;
        if Choice = LeftHPT.HPTInfo.Engine[Counter2] then
          begin
            Dialog_Kill := False;
            DialogScreen[1].Init(Concat('Correct',Chr(Random(10)+48),'.msg'));
            DialogScreen[1].Show(350,50);
            Ch := ReadKey;
            while Ch = 'h' do
              begin
                DialogScreen[1].Hide;
                DialogScreen[1].Kill;
                Dialog_Kill := True;
                DialogScreen[2].Hide;
                DialogScreen[2].Kill;
                HelpItem.GetHelp;
                Ch := ' ';
                DialogScreen[2].Init(Concat(Copy(
                        StudentModel.HPTArray[Counter1],1,5),'.nam'));
                DialogScreen[2].Show(72,50);
              end;
              if not Dialog_Kill then
                begin
                  DialogScreen[1].Hide;
                  DialogScreen[1].Kill;
                end;
          end
          else if Choice <> 'null' then
            begin
              Dialog_Kill := False;
              DialogScreen[1].Init(Concat('Wrong',Chr(Random(10)+48),'.msg'));
              DialogScreen[1].Show(350,50);
              ShowFeature(LeftHPT.HPTInfo.EngineInfo[Counter2]);
              if not Dialog_Kill then
                begin
                  DialogScreen[1].Hide;
```

```
                    DialogScreen[1].Kill;
                  end;
              end
            else Ch := #27;
        end;
for Counter2 := 1 to 1 do
    if (LeftHPT.HPTInfo.Fuslag[Counter2] <> '') and (Ch <> #27) then
        begin
          Dialog_Kill := False;
          WMenu.Init(Concat(Copy(LeftHPT.HPTInfo.FuslagInfo[Counter2],1,4),
                '.mnu'));
          Choice := WMenu.GetChoice;
          if Choice = LeftHPT.HPTInfo.Fuslag[Counter2] then
            begin
              Dialog_Kill := False;
              DialogScreen[1].Init(Concat('Correct',Chr(Random(10)+48),'.msg'));
              DialogScreen[1].Show(350,50);
              Ch := ReadKey;
              while Ch = 'h' do
                begin
                  DialogScreen[1].Hide;
                  DialogScreen[1].Kill;
                  Dialog_Kill := True;
                  DialogScreen[2].Hide;
                  DialogScreen[2].Kill;
                  HelpItem.GetHelp;
                  Ch := ' ';
                  DialogScreen[2].Init(Concat(Copy(
                        StudentModel.HPTArray[Counter1],1,5),'.nam'));
                  DialogScreen[2].Show(72,50);
                end;
              if not Dialog_Kill then
                begin
                  DialogScreen[1].Hide;
                  DialogScreen[1].Kill;
                end;
            end
            else if Choice <> 'null' then
              begin
                Dialog_Kill := False;
                DialogScreen[1].Init(Concat('Wrong',Chr(Random(10)+48),'.msg'));
                DialogScreen[1].Show(350,50);
                ShowFeature(LeftHPT.HPTInfo.FuslagInfo[Counter2]);
```

```
                if not Dialog_Kill then
                  begin
                    DialogScreen[1].Hide;
                    DialogScreen[1].Kill;
                  end;
              end
            else Ch := #27;
        end;
for Counter2 := 1 to 1 do
    if (LeftHPT.HPTInfo.Trot[Counter2] <> '') and (Ch <> #27) then
        begin
          WMenu.Init(Concat(Copy(LeftHPT.HPTInfo.TrotInfo[Counter2],1,4),
                '.mnu'));
          Choice := WMenu.GetChoice;
          SetColor(0);
          SetLineStyle(SolidLn,0,ThickWidth);
          for Liner := 1 to 18 do
              begin
                Line(391,242+Liner,560,242+Liner);
              end;
          if Choice = LeftHPT.HPTInfo.Trot[Counter2] then
            begin
              Dialog_Kill := False;
              DialogScreen[1].Init(Concat('Correct',Chr(Random(10)+48),'.msg'));
              DialogScreen[1].Show(350,50);
              Ch := ReadKey;
              while Ch = 'h' do
                begin
                  DialogScreen[1].Hide;
                  DialogScreen[1].Kill;
                  Dialog_Kill := True;
                  DialogScreen[2].Hide;
                  DialogScreen[2].Kill;
                  HelpItem.GetHelp;
                  Ch := ' ';
                  DialogScreen[2].Init(Concat(Copy(
                          StudentModel.HPTArray[Counter1],1,5),'.nam'));
                  DialogScreen[2].Show(72,50);
                end;
              if not Dialog_Kill then
                begin
                  DialogScreen[1].Hide;
                  DialogScreen[1].Kill;
```

```
                end;
            end
            else if Choice <> 'null' then
               begin
                  Dialog_Kill := False;
                  DialogScreen[1].Init(Concat('Wrong',Chr(Random(10)+48),'.msg'));
                  DialogScreen[1].Show(350,50);
                  ShowFeature(LeftHPT.HPTInfo.TrotInfo[Counter2]);
                  if not Dialog_Kill then
                     begin
                        DialogScreen[1].Hide;
                        DialogScreen[1].Kill;
                     end;
               end
            else Ch := #27;
      end;
for Counter2 := 1 to 1 do
   if (LeftHPT.HPTInfo.Mrot[Counter2] <> '') and (Ch <> #27) then
      begin
         WMenu.Init(Concat(Copy(LeftHPT.HPTInfo.MrotInfo[Counter2],1,4),
               '.mnu'));
         Choice := WMenu.GetChoice;
         SetColor(0);
         SetLineStyle(SolidLn,0,ThickWidth);
         for Liner := 1 to 18 do
            begin
               Line(391,242+Liner,560,242+Liner);
            end;
         if Choice = LeftHPT.HPTInfo.Mrot[Counter2] then
            begin
               Dialog_Kill := False;
               DialogScreen[1].Init(Concat('Correct',Chr(Random(10)+48),'.msg'));
               DialogScreen[1].Show(350,50);
               Ch := ReadKey;
               while Ch = 'h' do
                  begin
                     DialogScreen[1].Hide;
                     DialogScreen[1].Kill;
                     Dialog_Kill := True;
                     DialogScreen[2].Hide;
                     DialogScreen[2].Kill;
                     HelpItem.GetHelp;
                     Ch := ' ';
```

```
                DialogScreen[2].Init(Concat(Copy(
                        StudentModel.HPTArray[Counter1],1,5),'.nam'));
                DialogScreen[2].Show(72,50);
              end;
            if not Dialog_Kill then
              begin
                DialogScreen[1].Hide;
                DialogScreen[1].Kill;
              end;
          end
          else if Choice <> 'null' then
            begin
              Dialog_Kill := False;
              DialogScreen[1].Init(Concat('Wrong',Chr(Random(10)+48),'.msg'));
              DialogScreen[1].Show(350,50);
              ShowFeature(LeftHPT.HPTInfo.MrotInfo[Counter2]);
              if not Dialog_Kill then
                begin
                  DialogScreen[1].Hide;
                  DialogScreen[1].Kill;
                end;
            end
          else Ch := #27;
      end;
  for Counter2 := 1 to 1 do
    if (LeftHPT.HPTInfo.Ucag[Counter2] <> '') and (Ch <> #27) then
      begin
        WMenu.Init(Concat(Copy(LeftHPT.HPTInfo.UcagInfo[Counter2],1,4),
              '.mnu'));
        Choice := WMenu.GetChoice;
        SetColor(0);
        SetLineStyle(SolidLn,0,ThickWidth);
        for Liner := 1 to 18 do
          begin
            Line(391,242+Liner,560,242+Liner);
          end;
        if Choice = LeftHPT.HPTInfo.Ucag[Counter2] then
          begin
            Dialog_Kill := False;
            DialogScreen[1].Init(Concat('Correct',Chr(Random(10)+48),'.msg'));
            DialogScreen[1].Show(350,50);
            Ch := ReadKey;
            while Ch = 'h' do
```

```
                begin
                  DialogScreen[1].Hide;
                  DialogScreen[1].Kill;
                  Dialog_Kill := True;
                  DialogScreen[2].Hide;
                  DialogScreen[2].Kill;
                  HelpItem.GetHelp;
                  Ch := ' ';
                  DialogScreen[2].Init(Concat(Copy(
                          StudentModel.HPTArray[Counter1],1,5),'.nam'));
                  DialogScreen[2].Show(72,50);
                end;
              if not Dialog_Kill then
                begin
                  DialogScreen[1].Hide;
                  DialogScreen[1].Kill;
                end;
            end
            else if Choice <> 'null' then
              begin
                Dialog_Kill := False;
                DialogScreen[1].Init(Concat('Wrong',Chr(Random(10)+48),'.msg'));
                DialogScreen[1].Show(350,50);
                ShowFeature(LeftHPT.HPTInfo.UcagInfo[Counter2]);
                if not Dialog_Kill then
                  begin
                    DialogScreen[1].Hide;
                    DialogScreen[1].Kill;
                  end;
              end
            else Ch := #27;
        end;
for Counter2 := 1 to 1 do
    if (LeftHPT.HPTInfo.Hstl[Counter2] <> '') and (Ch <> #27) then
      begin
        WMenu.Init(Concat(Copy(LeftHPT.HPTInfo.HstlInfo[Counter2],1,4),
              '.mnu'));
        Choice := WMenu.GetChoice;
        SetColor(0);
        SetLineStyle(SolidLn,0,ThickWidth);
        for Liner := 1 to 18 do
            begin
              Line(391,242+Liner,560,242+Liner);
```

```
            end;
          if Choice = LeftHPT.HPTInfo.Hstl[Counter2] then
            begin
              Dialog_Kill := False;
              DialogScreen[1].Init(Concat('Correct',Chr(Random(10)+48),'.msg'));
              DialogScreen[1].Show(350,50);
              Ch := ReadKey;
              while Ch = 'h' do
                begin
                  DialogScreen[1].Hide;
                  DialogScreen[1].Kill;
                  Dialog_Kill := True;
                  DialogScreen[2].Hide;
                  DialogScreen[2].Kill;
                  HelpItem.GetHelp;
                  Ch := ' ';
                  DialogScreen[2].Init(Concat(Copy(
                                StudentModel.HPTArray[Counter1],1,5)
                               ,'.nam'));
                  DialogScreen[2].Show(72,50);
                end;
              if not Dialog_Kill then
                begin
                  DialogScreen[1].Hide;
                  DialogScreen[1].Kill;
                end;
            end
            else if Choice <> 'null' then
              begin
                Dialog_Kill := False;
                DialogScreen[1].Init(Concat('Wrong',Chr(Random(10)+48),'.msg'));
                DialogScreen[1].Show(350,50);
                ShowFeature(LeftHPT.HPTInfo.HstlInfo[Counter2]);
                if not Dialog_Kill then
                  begin
                    DialogScreen[1].Hide;
                    DialogScreen[1].Kill;
                  end;
              end
          else Ch := #27;
      end;
for Counter2 := 1 to 1 do
    if (LeftHPT.HPTInfo.Hstn[Counter2] <> '') and (Ch <> #27) then
```

```
if (LeftHPT.HPTInfo.Hstn[Counter2] <> '') and (Ch <> #27) then
  begin
    WMenu.Init(Concat(Copy(LeftHPT.HPTInfo.HstnInfo[Counter2],1,4),
          '.mnu'));
    Choice := WMenu.GetChoice;
    SetColor(0);
    SetLineStyle(SolidLn,0,ThickWidth);
    for Liner := 1 to 18 do
      begin
        Line(391,242+Liner,560,242+Liner);
      end;
    if Choice = LeftHPT.HPTInfo.Hstn[Counter2] then
      begin
        Dialog_Kill := False;
        DialogScreen[1].Init(Concat('Correct',Chr(Random(10)+48),'.msg'));
        DialogScreen[1].Show(350,50);
        Ch := ReadKey;
        while Ch = 'h' do
          begin
            DialogScreen[1].Hide;
            DialogScreen[1].Kill;
            Dialog_Kill := True;
            DialogScreen[2].Hide;
            DialogScreen[2].Kill;
            HelpItem.GetHelp;
            Ch := ' ';
            DialogScreen[2].Init(Concat(Copy(
                    StudentModel.HPTArray[Counter1],1,5),'.nam'));
            DialogScreen[2].Show(72,50);
          end;
        if not Dialog_Kill then
          begin
            DialogScreen[1].Hide;
            DialogScreen[1].Kill;
          end;
      end
      else if Choice <> 'null' then
        begin
          Dialog_Kill := False;
          DialogScreen[1].Init(Concat('Wrong',Chr(Random(10)+48),'.msg'));
          DialogScreen[1].Show(350,50);
          ShowFeature(LeftHPT.HPTInfo.HstnInfo[Counter2]);
          if not Dialog_Kill then
```

```
                    begin
                      DialogScreen[1].Hide;
                      DialogScreen[1].Kill;
                    end;
                end
              else Ch := #27;
          end;
    DialogScreen[2].Hide;
    DialogScreen[2].Kill;
end;

procedure HandleCloseIntermediate;
begin
    SetColor(14);
    if Copy(Choice,5,1) = ' ' then
      RightHPT.Show(344,225,Concat(Copy(Choice,1,4),'_',
                Copy(StudentModel.HPTArray[Counter1],6,3)))
    else
      RightHPT.Show(344,225,Concat(Copy(Choice,1,5),
                Copy(StudentModel.HPTArray[Counter1],6,3)));
    DialogScreen[1].Init('Close.msg');
    DialogScreen[1].Show(205,60);
    Ch := Readkey;
    if Ch = 'h' then
      HelpItem.GetHelp;
    DialogScreen[1].Hide;
    DialogScreen[1].Kill;
    if Ch = #27 then
      Exit;
    DialogScreen[1].Init('Compare.msg');
    DialogScreen[1].Show(205,60);
    Ch := ReadKey;
    DialogScreen[1].Hide;
    DialogScreen[2].Init(Concat(Copy(StudentModel.HPTArray[Counter1],1,5),'.nam'));
    if Copy(Choice,5,1) = ' ' then
      DialogScreen[3].Init(Concat(Copy(Choice,1,4),'_','.nam'))
    else DialogScreen[3].Init(Concat(Copy(Choice,1,5),'.nam'));
      DialogScreen[2].Show(72,50);
    DialogScreen[3].Show(350,50);
    Ch := ReadKey;
    for Counter2 := 1 to 1 do
        if (LeftHPT.HPTInfo.Wings[Counter2] <> '') and (Ch <> #27) and
          (RightHPT.HPTInfo.Wings[Counter2] <> '') then
```

```
        RightHPT.HPTInfo.Wings[Counter2] then
          begin
            DialogScreen[4].Init(Concat(LeftHPT.HPTInfo.WingsInfo[Counter2],
                        '.msg'));
            DialogScreen[5].Init(Concat(RightHPT.HPTInfo.WingsInfo[Counter2],
                        '.msg'));
            DialogScreen[4].Show(10,30);
            DialogScreen[5].Show(430,30);
            Ch := ReadKey;
            DialogScreen[5].Hide;
            DialogScreen[4].Hide;
            DialogScreen[5].Kill;
            DialogScreen[4].Kill;
          end;
for Counter2 := 1 to 1 do
   if (LeftHPT.HPTInfo.Engine[Counter2] <> '') and (Ch <> #27) and
     (RightHPT.HPTInfo.Engine[Counter2] <> '') then
        if LeftHPT.HPTInfo.Engine[Counter2] <>
          RightHPT.HPTInfo.Engine[Counter2] then
          begin
            DialogScreen[4].Init(Concat(LeftHPT.HPTInfo.EngineInfo[Counter2],
                  '.msg'));
            DialogScreen[5].Init(Concat(RightHPT.HPTInfo.EngineInfo[
                  Counter2],'.msg'));
            DialogScreen[4].Show(10,30);
            DialogScreen[5].Show(430,30);
            Ch := ReadKey;
            DialogScreen[5].Hide;
            DialogScreen[4].Hide;
            DialogScreen[5].Kill;
            DialogScreen[4].Kill;
          end;
for Counter2 := 1 to 1 do
   if (LeftHPT.HPTInfo.Fuslag[Counter2] <> '') and (Ch <> #27) and
     (RightHPT.HPTInfo.Fuslag[Counter2] <> '') then
        if LeftHPT.HPTInfo.Fuslag[Counter2] <>
          RightHPT.HPTInfo.Fuslag[Counter2] then
          begin
            DialogScreen[4].Init(Concat(
                  LeftHPT.HPTInfo.FuslagInfo[Counter2],'.msg'));
            DialogScreen[5].Init(Concat(
                  RightHPT.HPTInfo.FuslagInfo[Counter2],'.msg'));
            DialogScreen[4].Show(10,30);
```

```
                    RightHPT.HPTInfo.FuslagInfo[Counter2],'.msg'));
            DialogScreen[4].Show(10,30);
            DialogScreen[5].Show(430,30);
            Ch := ReadKey;
            DialogScreen[5].Hide;
            DialogScreen[4].Hide;
            DialogScreen[5].Kill;
            DialogScreen[4].Kill;
          end;
for Counter2 := 1 to 1 do
  if (LeftHPT.HPTInfo.Trot[Counter2] <> '') and (Ch <> #27) and
    (RightHPT.HPTInfo.Trot[Counter2] <> '') then
      if LeftHPT.HPTInfo.Trot[Counter2] <>
        RightHPT.HPTInfo.Trot[Counter2] then
        begin
          DialogScreen[4].Init(Concat(
                    LeftHPT.HPTInfo.TrotInfo[Counter2],'.msg'));
          DialogScreen[5].Init(Concat(
                    RightHPT.HPTInfo.TrotInfo[Counter2],'.msg'));
          DialogScreen[4].Show(10,30);
          DialogScreen[5].Show(430,30);
          Ch := ReadKey;
          DialogScreen[5].Hide;
          DialogScreen[4].Hide;
          DialogScreen[5].Kill;
          DialogScreen[4].Kill;
        end;
for Counter2 := 1 to 1 do
  if (LeftHPT.HPTInfo.Mrot[Counter2] <> '') and (Ch <> #27) and
    (RightHPT.HPTInfo.Mrot[Counter2] <> '') then
      if LeftHPT.HPTInfo.Mrot[Counter2] <>
        RightHPT.HPTInfo.Mrot[Counter2] then
        begin
          DialogScreen[4].Init(Concat(
                    LeftHPT.HPTInfo.MrotInfo[Counter2],'.msg'));
          DialogScreen[5].Init(Concat(
                    RightHPT.HPTInfo.MrotInfo[Counter2],'.msg'));
          DialogScreen[4].Show(10,30);
          DialogScreen[5].Show(430,30);
          Ch := ReadKey;
          DialogScreen[5].Hide;
          DialogScreen[4].Hide;
          DialogScreen[5].Kill;
```

```
                DialogScreen[4].Kill;
              end;
for Counter2 := 1 to 1 do
   if (LeftHPT.HPTInfo.Ucag[Counter2] <> '') and (Ch <> #27) and
      (RightHPT.HPTInfo.Ucag[Counter2] <> '') then
         if LeftHPT.HPTInfo.Ucag[Counter2] <>
            RightHPT.HPTInfo.Ucag[Counter2] then
            begin
               DialogScreen[4].Init(Concat(
                        LeftHPT.HPTInfo.UcagInfo[Counter2],'.msg'));
               DialogScreen[5].Init(Concat(
                        RightHPT.HPTInfo.UcagInfo[Counter2],'.msg'));
               DialogScreen[4].Show(10,30);
               DialogScreen[5].Show(430,30);
               Ch := ReadKey;
               DialogScreen[5].Hide;
               DialogScreen[4].Hide;
               DialogScreen[5].Kill;
               DialogScreen[4].Kill;
            end;
for Counter2 := 1 to 1 do
   if (LeftHPT.HPTInfo.Hstl[Counter2] <> '') and (Ch <> #27) and
      (RightHPT.HPTInfo.Hstl[Counter2] <> '') then
         if LeftHPT.HPTInfo.Hstl[Counter2] <>
            RightHPT.HPTInfo.Hstl[Counter2] then
            begin
               DialogScreen[4].Init(Concat(
                        LeftHPT.HPTInfo.HstlInfo[Counter2],'.msg'));
               DialogScreen[5].Init(Concat(
                        RightHPT.HPTInfo.HstlInfo[Counter2],'.msg'));
               DialogScreen[4].Show(10,30);
               DialogScreen[5].Show(430,30);
               Ch := ReadKey;
               DialogScreen[5].Hide;
               DialogScreen[4].Hide;
               DialogScreen[5].Kill;
               DialogScreen[4].Kill;
            end;
for Counter2 := 1 to 1 do
   if (LeftHPT.HPTInfo.Hstn[Counter2] <> '') and (Ch <> #27) and
      (RightHPT.HPTInfo.Hstn[Counter2] <> '') then
         if LeftHPT.HPTInfo.Hstn[Counter2] <>
            RightHPT.HPTInfo.Hstn[Counter2] then
```

```
                begin
                  DialogScreen[4].Init(Concat(
                        LeftHPT.HPTInfo.HstnInfo[Counter2],'.msg'));
                  DialogScreen[5].Init(Concat(
                        RightHPT.HPTInfo.HstnInfo[Counter2],'.msg'));
                  DialogScreen[4].Show(10,30);
                  DialogScreen[5].Show(430,30);
                  Ch := ReadKey;
                  DialogScreen[5].Hide;
                  DialogScreen[4].Hide;
                  DialogScreen[5].Kill;
                  DialogScreen[4].Kill;
                end;
    DialogScreen[1].Kill;
    DialogScreen[2].Hide;
    DialogScreen[2].Kill;
    DialogScreen[3].Hide;
    DialogScreen[3].Kill;
    RightHPT.Hide;
end;

begin {ReviewIntermediate}
   MaxNum := 150;
   Ch := #13;
   Done := False;
   DialogScreen[1].Init('Return.msg');
   DialogScreen[1].Show(205,60);
   Ch := ReadKey;
   while Ch = 'h' do
      begin
         HelpItem.GetHelp;
         Ch := ReadKey;
      end;
   DialogScreen[1].Hide;
   DialogScreen[1].Kill;
   if Ch = #27 then Exit;
   DialogScreen[1].Init('RevInt.msg');
   DialogScreen[1].Show(205,60);
   Ch := ReadKey;
   while Ch = 'h' do
      begin
         HelpItem.GetHelp;
         Ch := ReadKey;
```

```
  end;
DialogScreen[1].Hide;
DialogScreen[1].Kill;
if Ch = #27 then Exit;
Counter1 := StudentModel.GetEntry(MaxNum);
while (Counter1 <> 0) and (Done = False) do
  begin
    LeftHPT.Init(StudentModel.HPTArray[Counter1]);
    WMenu.Init('WETFUR.mnu');
    CorrectAnswer := False;
    CloseAnswer := False;
    LeftHPT.Show(72,225,StudentModel.HPTArray[Counter1]);
    Choice := WMenu.GetChoice;
    if Choice = 'null' then
      begin
        LeftHPT.Hide;
        LeftHPT.Kill;
        Exit;
      end;
    if Copy(Choice,1,5) =
      Copy(LeftHPT.HPTInfo.HelicopterName,1,5) then
      HandleCorrectIntermediate
    else
      begin
        if Copy(Choice,5,1) = ' ' then
                RightHPT.Init(Concat(Copy(Choice,1,4),'_',
                    Copy(StudentModel.HPTArray[Counter1],6,3)))
        else
          RightHPT.Init(Concat(Copy(Choice,1,5),
                    Copy(StudentModel.HPTArray[Counter1],6,3)));
        if CompareHelicopter >= 0.7 then
          begin
            HandleCloseIntermediate;
            RightHPT.Hide;
            RightHPT.Kill;
          end
        else
          begin
            HandleWrongIntermediate;
            if StudentModel.AddEntry(
            StudentModel.HPTArray[Counter1],MaxNum) = False then
              Done := True;
          end;
```

```
            end;
          GoToXY(1,1);
          LeftHPT.Hide;
          LeftHPT.Kill;
          if Ch = #27 then Exit;
            Counter1 := StudentModel.GetEntry(MaxNum);
        end;
    Done := True;
end;

procedure ReviewExpert;
var
    Feature : array [1..16] of Name;
begin
    MaxNum := 150;
    Ch := #13;
    Done := False;
    DialogScreen[1].Init('Return.msg');
    DialogScreen[1].Show(205,60);
    Ch := ReadKey;
    while Ch = 'h' do
        begin
            HelpItem.GetHelp;
            Ch := ReadKey;
        end;
    DialogScreen[1].Hide;
    DialogScreen[1].Kill;
    if Ch = #27 then Exit;
    DialogScreen[1].Init('RevExp.msg');
    DialogScreen[1].Show(205,60);
    Ch := ReadKey;
    while Ch = 'h' do
        begin
            HelpItem.GetHelp;
            Ch := ReadKey;
        end;
    DialogScreen[1].Hide;
    DialogScreen[1].Kill;
    if Ch = #27 then Exit;
    Counter1 := StudentModel.GetEntry(MaxNum);
    while (Counter1 <> 0) and (Done = False) do
        begin
            Dialog_Kill := False;
```

```
Counter3 := 1;
LeftHPT.Init(StudentModel.HPTArray[Counter1]);
WMenu.Init('WETFUR.mnu');
for Counter2 := 1 to 1 do
   if LeftHPT.HPTInfo.WingsInfo[Counter2] <> '' then
      begin
         Feature[Counter3] := LeftHPT.HPTInfo.WingsInfo[Counter2];
         Counter3 := Counter3 + 1;
      end;
for Counter2 := 1 to 1 do
   if LeftHPT.HPTInfo.EngineInfo[Counter2] <> '' then
      begin
         Feature[Counter3] := LeftHPT.HPTInfo.EngineInfo[Counter2];
         Counter3 := Counter3 + 1;
      end;
for Counter2 := 1 to 1 do
   if LeftHPT.HPTInfo.FuslagInfo[Counter2] <> '' then
      begin
         Feature[Counter3] := LeftHPT.HPTInfo.FuslagInfo[Counter2];
         Counter3 := Counter3 + 1;
      end;
for Counter2 := 1 to 1 do
   if LeftHPT.HPTInfo.TrotInfo[Counter2] <> '' then
      begin
         Feature[Counter3] := LeftHPT.HPTInfo.TrotInfo[Counter2];
         Counter3 := Counter3 + 1;
      end;
for Counter2 := 1 to 1 do
   if LeftHPT.HPTInfo.MrotInfo[Counter2] <> '' then
      begin
         Feature[Counter3] := LeftHPT.HPTInfo.MrotInfo[Counter2];
         Counter3 := Counter3 + 1;
      end;
for Counter2 := 1 to 1 do
   if LeftHPT.HPTInfo.UcagInfo[Counter2] <> '' then
      begin
         Feature[Counter3] := LeftHPT.HPTInfo.UcagInfo[Counter2];
         Counter3 := Counter3 + 1;
      end;
for Counter2 := 1 to 1 do
   if LeftHPT.HPTInfo.HstlInfo[Counter2] <> '' then
      begin
         Feature[Counter3] := LeftHPT.HPTInfo.HstnInfo[Counter2];
```

```
            Counter3 := Counter3 + 1;
         end;
for Counter2 := 1 to 1 do
    if LeftHPT.HPTInfo.HstnInfo[Counter2] <> '' then
         begin
            Feature[Counter3] := LeftHPT.HPTInfo.HstlInfo[Counter2];
            Counter3 := Counter3 + 1;
         end;
Counter2 := 1;
Ch := #8;
while Ch <> #13 do
   begin
      ShowFeature(Feature[Counter2]);
      if (Ch = '-') and (Counter2 > 1) then
         Counter2 := Counter2 - 1
      else if (Ch = '-') and (Counter2 = 1) then
         Counter2 := Counter3 - 1
      else if (Ch = '+') and (Counter2 < Counter3 - 1) then
         Counter2 := Counter2 + 1
      else if (Ch = '+') and (Counter2 = Counter3 - 1) then
         Counter2 := 1
      else if Ch = #27 then Exit
      else
         begin
            Sound(440);
            Delay(200);
            NoSound;
         end;
   end;
Choice := WMenu.GetChoice;
if Choice = 'null' then
   Exit;
DialogScreen[2].Init(Concat(Copy(
            StudentModel.HPTArray[Counter1],1,5),'.nam'));
LeftHPT.Show(72,225,StudentModel.HPTArray[Counter1]);
DialogScreen[2].Show(72,50);
if Copy(Choice,1,5) = Copy(LeftHPT.HPTInfo.HelicopterName,1,5) then
   begin
      DialogScreen[1].Init(Concat('Correct',Chr(Random(10)+48),'.msg'));
      DialogScreen[1].Show(350,50);
      Ch := ReadKey;
      while Ch = 'h' do
         begin
```

```
            DialogScreen[1].Hide;
            DialogScreen[1].Kill;
            Dialog_Kill := True;
            DialogScreen[2].Hide;
            DialogScreen[2].Kill;
            HelpItem.GetHelp;
            Ch := #8;
            DialogScreen[2].Init(Concat(Copy(
                        StudentModel.HPTArray[Counter1],1,5),'.nam'));
            DialogScreen[2].Show(72,50);
            Ch := ReadKey;
          end;
        if not Dialog_Kill then
          begin
            DialogScreen[1].Hide;
            DialogScreen[1].Kill;
          end;
        StudentModel.HPTArray[Counter1] := '';
      end
      else
        begin
          DialogScreen[1].Init(Concat('Wrong',Chr(Random(10)+48),'.msg'));
          DialogScreen[1].Show(350,50);
          Ch := ReadKey;
          if Copy(Choice,5,1) = ' ' then
            begin
              RightHPT.Init(Concat(Copy(Choice,1,4),'_',
                        Copy(StudentModel.HPTArray[Counter1],6,3)));
              DialogScreen[3].Init(Concat(Copy(Choice,1,4),'_','.nam'));
              RightHPT.Show(344,225,Concat(Copy(Choice,1,4),'_',
                         Copy(StudentModel.HPTArray[Counter1],6,3)));
            end
          else
            begin
              RightHPT.Init(Concat(Copy(Choice,1,5),
                        Copy(StudentModel.HPTArray[Counter1],6,3)));
              DialogScreen[3].Init(Concat(Copy(Choice,1,5),'.nam'));
              RightHPT.Show(344,225,Concat(Copy(Choice,1,5),
                         Copy(StudentModel.HPTArray[Counter1],6,3)));
            end;
          DialogScreen[1].Hide;
          DialogScreen[1].Kill;
          DialogScreen[3].Show(350,50);
```

```
                Ch := ReadKey;
                DialogScreen[3].Hide;
                DialogScreen[3].Kill;
                RightHPT.Hide;
                RightHPT.Kill;
                if StudentModel.AddEntry(
                        StudentModel.HPTArray[Counter1],MaxNum) = False then
                   Done := True;
             end;
          GoToXY(1,1);
          DialogScreen[2].Hide;
          DialogScreen[2].Kill;
          LeftHPT.Hide;
          LeftHPT.Kill;
          if Ch = #27 then Exit;
             Counter1 := StudentModel.GetEntry(MaxNum);
       end;
    Done := True;
end;

procedure TestIntermediate;
begin
    Ch := #13;
    Done := False;
    MaxNum := 150;
    DialogScreen[1].Init('Return.msg');
    DialogScreen[1].Show(205,60);
    Ch := ReadKey;
    while Ch = 'h' do
       begin
          HelpItem.GetHelp;
          Ch := ReadKey;
       end;
    DialogScreen[1].Hide;
    DialogScreen[1].Kill;
    if Ch = #27 then Exit;
    DialogScreen[1].Init('TestInt.msg');
    DialogScreen[1].Show(205,60);
    Ch := ReadKey;
    while Ch = 'h' do
       begin
          HelpItem.GetHelp;
          Ch := ReadKey;
```

```
    end;
  DialogScreen[1].Hide;
  DialogScreen[1].Kill;
  if Ch = #27 then Exit;
  Counter1 := StudentModel.GetEntry(MaxNum);
  while Counter1 <> 0 do
    begin
      if StudentModel.HPTArray[Counter1] <> '' then
        begin
          LeftHPT.Init(StudentModel.HPTArray[Counter1]);
          LeftHPT.Show(72,225,StudentModel.HPTArray[Counter1]);
          WMenu.Init('WETFUR.mnu');
          Choice := WMenu.GetChoice;
          if Choice = 'null' then
            begin
              LeftHPT.Hide;
              LeftHPT.Kill;
              Exit;
            end;
          if Choice <> LeftHPT.HPTInfo.HelicopterName then
            begin
              Inc(StudentModel.NumMissed);
              Sound(100);
              Delay(200);
              NoSound;
            end;
          Inc(StudentModel.NumShown);
          LeftHPT.Hide;
          LeftHPT.Kill;
        end;
      GoToXY(1,1);
      StudentModel.HPTArray[Counter1] := '';
      Counter1 := StudentModel.GetEntry(MaxNum);
    end;
  Done := True;
end;

procedure TestExpert;
var
   Feature : array [1..16] of Name;
begin
  MaxNum := 150;
  Ch := #13;
```

```
Done := False;
DialogScreen[1].Init('Return.msg');
DialogScreen[1].Show(205,60);
Ch := ReadKey;
while Ch = 'h' do
  begin
    HelpItem.GetHelp;
    Ch := ReadKey;
  end;
DialogScreen[1].Hide;
DialogScreen[1].Kill;
if Ch = #27 then Exit;
DialogScreen[1].Init('TestExp.msg');
DialogScreen[1].Show(205,60);
Ch := ReadKey;
while Ch = 'h' do
  begin
    HelpItem.GetHelp;
    Ch := ReadKey;
  end;
DialogScreen[1].Hide;
DialogScreen[1].Kill;
if Ch = #27 then Exit;
Counter1 := StudentModel.GetEntry(MaxNum);
while Counter1 <> 0 do
  begin
    if StudentModel.HPTArray[Counter1] <> '' then
      begin
        LeftHPT.Init(StudentModel.HPTArray[Counter1]);
        WMenu.Init('WETFUR.mnu');
        Counter3 := 1;
        for Counter2 := 1 to 1 do
          if LeftHPT.HPTInfo.WingsInfo[Counter2] <> '' then
            begin
              Feature[Counter3] := LeftHPT.HPTInfo.WingsInfo[Counter2];
              Counter3 := Counter3 + 1;
            end;
        for Counter2 := 1 to 1 do
          if LeftHPT.HPTInfo.EngineInfo[Counter2] <> '' then
            begin
              Feature[Counter3] := LeftHPT.HPTInfo.EngineInfo[Counter2];
              Counter3 := Counter3 + 1;
            end;
```

```
for Counter2 := 1 to 1 do
   if LeftHPT.HPTInfo.FuslagInfo[Counter2] <> '' then
      begin
        Feature[Counter3] := LeftHPT.HPTInfo.FuslagInfo[Counter2];
        Counter3 := Counter3 + 1;
      end;
for Counter2 := 1 to 1 do
   if LeftHPT.HPTInfo.TrotInfo[Counter2] <> '' then
      begin
        Feature[Counter3] := LeftHPT.HPTInfo.TrotInfo[Counter2];
        Counter3 := Counter3 + 1;
      end;
for Counter2 := 1 to 1 do
   if LeftHPT.HPTInfo.MrotInfo[Counter2] <> '' then
      begin
        Feature[Counter3] := LeftHPT.HPTInfo.MrotInfo[Counter2];
        Counter3 := Counter3 + 1;
      end;
for Counter2 := 1 to 1 do
   if LeftHPT.HPTInfo.UcagInfo[Counter2] <> '' then
      begin
        Feature[Counter3] := LeftHPT.HPTInfo.UcagInfo[Counter2];
        Counter3 := Counter3 + 1;
      end;
for Counter2 := 1 to 1 do
   if LeftHPT.HPTInfo.HstnInfo[Counter2] <> '' then
      begin
        Feature[Counter3] := LeftHPT.HPTInfo.HstnInfo[Counter2];
        Counter3 := Counter3 + 1;
      end;
for Counter2 := 1 to 1 do
   if LeftHPT.HPTInfo.HstlInfo[Counter2] <> '' then
      begin
        Feature[Counter3] := LeftHPT.HPTInfo.HstlInfo[Counter2];
        Counter3 := Counter3 + 1;
      end;
Counter2 := 1;
Ch := #8;
while Ch <> #13 do
  begin
    ShowFeature(Feature[Counter2]);
      if (Ch = '-') and (Counter2 > 1) then
        Counter2 := Counter2 - 1
```

```
                else if (Ch = '-') and (Counter2 = 1) then
                  Counter2 := Counter3 - 1
                else if (Ch = '+') and (Counter2 < Counter3 - 1) then
                  Counter2 := Counter2 + 1
                else if (Ch = '+') and (Counter2 = Counter3 - 1) then
                  Counter2 := 1
                else if Ch = #27 then Exit
                else
                  begin
                    Sound(440);
                    Delay(200);
                    NoSound;
                  end;
              end;
            Choice := WMenu.GetChoice;
            if Choice = 'null' then
              Exit;
            LeftHPT.Show(72,225,StudentModel.HPTArray[Counter1]);
            if Choice <> LeftHPT.HPTInfo.HelicopterName then
              begin
                Inc(StudentModel.NumMissed);
                Sound(100);
                Delay(200);
                NoSound;
              end;
            Inc(StudentModel.NumShown);
            Ch := ReadKey;
            while Ch = 'h' do
              begin
                HelpItem.GetHelp;
                Ch := ReadKey;
              end;
            LeftHPT.Hide;
            LeftHPT.Kill;
            if Ch = #27 then Exit;
              GoToXY(1,1);
              StudentModel.HPTArray[Counter1] := '';
              Counter1 := StudentModel.GetEntry(MaxNum);
          end;
      end;
    Done := True;
end;
```

```
procedure EvaluateStudent;
var F : Text;
begin
  if (StudentModel.Mode = 'Teach') and
    (StudentModel.NumShown >= MaxNum) then
    begin
      StudentModel.Update(StudentModel.StudentName,'Review',
            StudentModel.Level,StudentModel.TestScore);
      DialogScreen[1].Init('Advance1.msg');
      DialogScreen[1].Show(205,60);
      Ch := ReadKey;
      DialogScreen[1].Hide;
      DialogScreen[1].Kill;
    end
  else if (StudentModel.Mode = 'Review') and (Done = True) then
    begin
      if StudentModel.GetEntry(MaxNum + 150) = 0 then
        begin
          if StudentModel.Level = 'Novice' then
            begin
              StudentModel.Update(StudentModel.StudentName,'Teach',
                    'Intermediate',StudentModel.TestScore);
              DialogScreen[1].Init('Advance2.msg');
              DialogScreen[1].Show(205,60);
              Ch := ReadKey;
              DialogScreen[1].Hide;
              DialogScreen[1].Kill;
            end
          else
            begin
              StudentModel.Update(StudentModel.StudentName,'Test',
                    StudentModel.Level,StudentModel.TestScore);
              DialogScreen[1].Init('Advance1.msg');
              DialogScreen[1].Show(205,60);
              Ch := ReadKey;
              DialogScreen[1].Hide;
              DialogScreen[1].Kill;
            end
        end
        else
          for Counter1 := 1 to MaxNum do
            begin
```

```
                    StudentModel.HPTArray[Counter1] :=
StudentModel.MissedArray[Counter1];
                    StudentModel.MissedArray[Counter1] := '';
                    StudentModel.NumShown := 1;
                    StudentModel.NumMissed := 0;
                  end
      end
    else if (StudentModel.Mode = 'Test') and (Done = True) then
      begin
        if StudentModel.NumMissed = 0 then
          begin
            if StudentModel.Level = 'Expert' then
              begin
                Assign(F,'HallFame.rec');
                Append(F);
                Writeln(F,StudentModel.StudentName);
                Close(F);
                StudentModel.Kill
              end
            else
              begin
                DialogScreen[1].Init('Great.msg');
                DialogScreen[1].Show(205,60);
                StudentModel.Update(StudentModel.StudentName,
                                    'Review','Expert',100);
                Ch := ReadKey;
                DialogScreen[1].Hide;
                DialogScreen[1].Kill;
              end
          end
        else
          begin
            Score:= Round(100*(1-StudentModel.NumMissed/
                              StudentModel.NumShown-1)));
            DialogScreen[1].Init('Score.msg');
            DialogScreen[1].Show(205,60);
            SetColor(12);
            OutTextXY(315,160,Chr(Score div 10 + 48));
            OutTextXY(325,160,Chr(Score mod 10 + 48));
            SetColor(0);
            Ch := ReadKey;
            DialogScreen[1].Hide;
            DialogScreen[1].Kill;
```

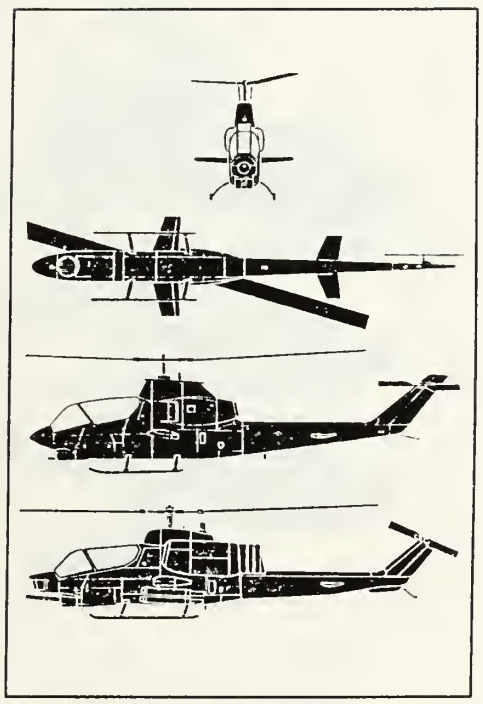
```
if Score >= 90 then
  begin
    if StudentModel.Level = 'Expert' then
      begin
        DialogScreen[1].Init('Outst.msg');
        DialogScreen[1].Show(205,60);
        StudentModel.Update(StudentModel.StudentName,
                            'Test','Expert',Score);
        Ch := ReadKey;
        DialogScreen[1].Hide;
        DialogScreen[1].Kill;
      end
    else
      begin
        DialogScreen[1].Init('Good.msg');
        DialogScreen[1].Show(205,60);
        Ch := ReadKey;
        DialogScreen[1].Hide;
        DialogScreen[1].Kill;
        if StudentModel.Level = 'Novice' then
          StudentModel.Update(StudentModel.StudentName,'Teach',
                'Intermediate',Score)
        else StudentModel.Update(StudentModel.StudentName,Review',
                    'Expert',Score);
      end
  end
  else if Score >= 80 then
    begin
      DialogScreen[1].Init('Fair.msg');
      DialogScreen[1].Show(205,60);
      Ch := ReadKey;
      DialogScreen[1].Hide;
      DialogScreen[1].Kill;
      StudentModel.Update(StudentModel.StudentName,'Test',
                    StudentModel.Level,Score);
    end
  else if Score >= 70 then
    begin
      DialogScreen[1].Init('Poor.msg');
      DialogScreen[1].Show(205,60);
      Ch := ReadKey;
      DialogScreen[1].Hide;
      DialogScreen[1].Kill;
```

```
                    StudentModel.Update(StudentModel.StudentName,'Review',
                            StudentModel.Level,Score);
                  end
                else
                  begin
                    DialogScreen[1].Init('Fail.msg');
                    DialogScreen[1].Show(205,60);
                    Ch := ReadKey;
                    DialogScreen[1].Hide;
                    DialogScreen[1].Kill;
                    if StudentModel.Level = 'Expert' then
                      StudentModel.Update(StudentModel.StudentName,'Review',
                            'Intermediate',Score)
                    else StudentModel.Update(StudentModel.StudentName,'Review',
                                        'Novice',Score);
                  end;
            end;
      end;
end;

procedure Tutor;
begin {tutor}
  if not StudentModel.Get then
    Diagnose;
  Ch := 'C';
  while UpCase(Ch) = 'C' do
    begin
      SetColor(12);
      SetTextJustify(CenterText,CenterText);
      OutTextXY(320,425,Concat(StudentModel.Mode,'/',StudentModel.Level));
      if StudentModel.Level = 'Novice' then
        begin
          if StudentModel.Mode = 'Teach' then Teach
          else if StudentModel.Mode = 'Review' then ReviewNovice;
        end
      else if StudentModel.Level = 'Intermediate' then
        begin
          if StudentModel.Mode = 'Teach' then Teach
          else if StudentModel.Mode = 'Review' then
            ReviewIntermediate
          else if StudentModel.Mode = 'Test' then
            TestIntermediate;
        end
```

```
        else if StudentModel.Level = 'Expert' then
          begin
            if StudentModel.Mode = 'Review' then
              ReviewExpert
            else if StudentModel.Mode = 'Test' then
              TestExpert;
          end;
        SetColor(0);
        SetTextJustify(CenterText,CenterText);
        OutTextXY(320,425,Concat(StudentModel.Mode,'/',StudentModel.Level));
        EvaluateStudent;
        DialogScreen[1].Init('Contin.msg');
        DialogScreen[1].Show(220,75);
        Ch := ReadKey;
        DialogScreen[1].Hide;
        DialogScreen[1].Kill;
      end;
    StudentModel.Save;
    Mainmenu.MInit('main.mnu');
end; {tutor}

end.
```

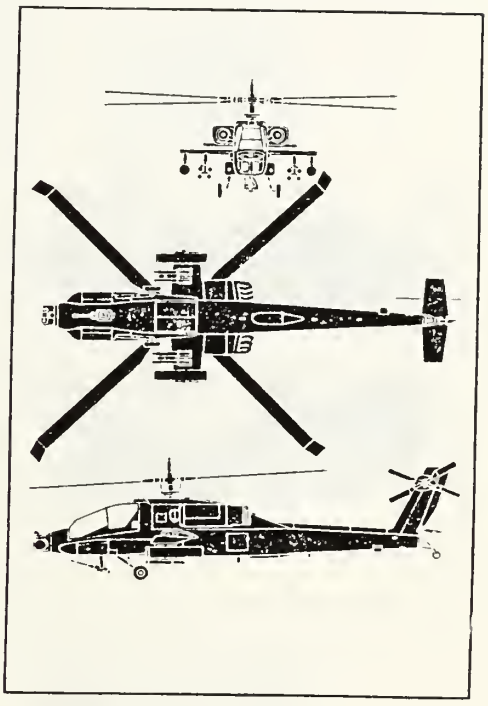
# APPENDIX C - HELICOPTER IMAGES

AH-1 HueyCobra



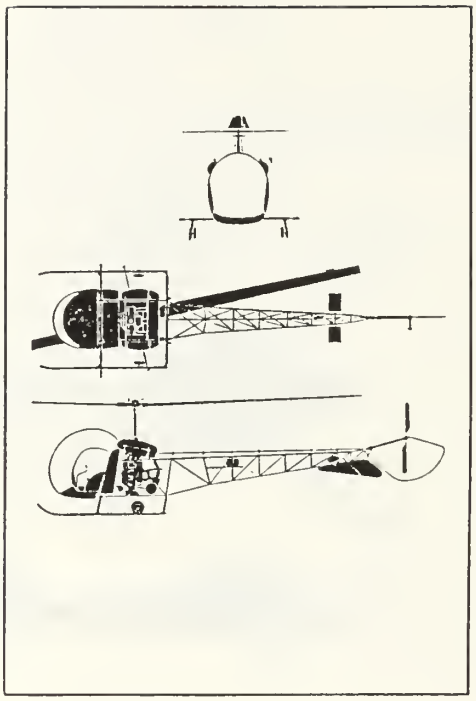
Wings: in stub form
Engine: single turboshaft
Horizontal Stabilizer:
mid-mounted on tail boom
and full
Fuselage: fair tail boom
Undercarriage: skid
Main Rotor: single
Tail Rotor Placement:
external

AH-64A Apache



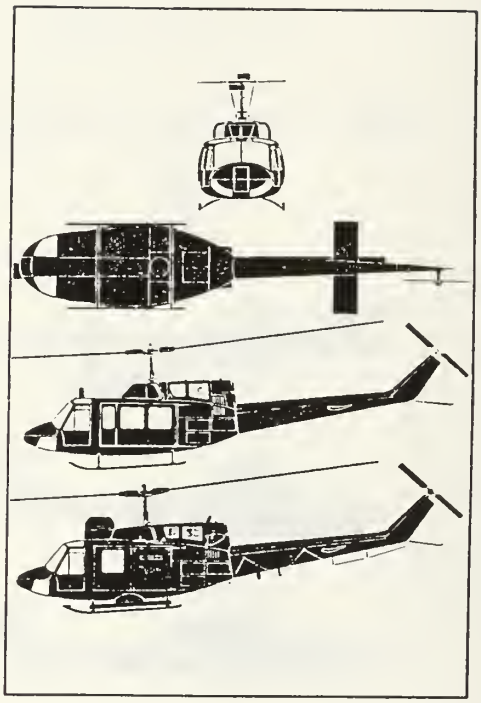
Wings: in stub form
Engine: single turboshaft
Horizontal Stabilizer:
end-mounted on tail boom
and full
Fuselage: fair tail boom
Undercarriage: wheeled
Main Rotor: Single
Tail Rotor Placement:
external

Bell 47G2



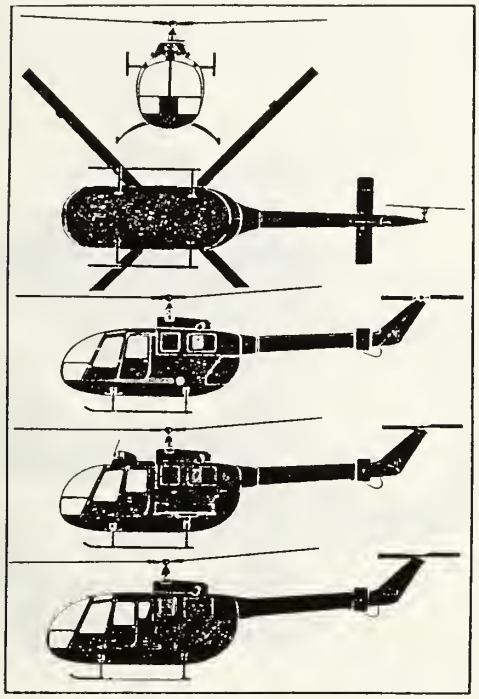
Wings: no
Engine: single piston
Horizontal Stabilizer:
end-mounted on tail boom
and full
Fuselage: open tail boom
Undercarriage: skid
Main Rotor: single
Tail Rotor Placement:
external

Bell 212



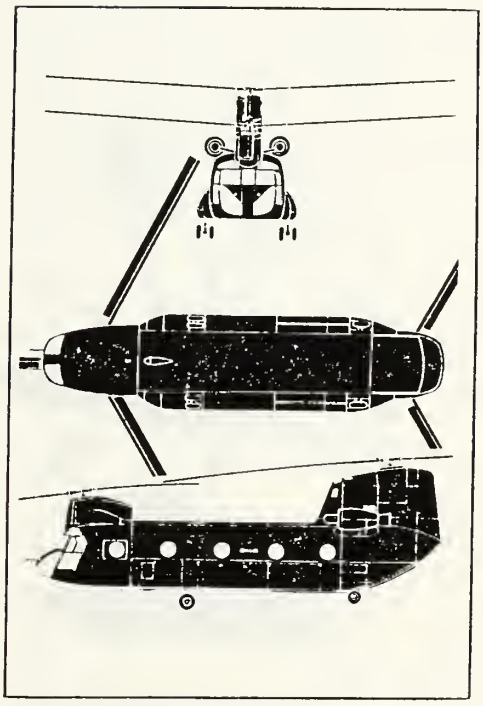
Wings: no
Engine: single turboshaft
Horizontal Stabilizer:
mid-mounted on tail boom
and full
Fuselage: fair tail boom
Undercarriage: skid
Main Rotor: single
Tail Rotor Placement:
external

BO 105



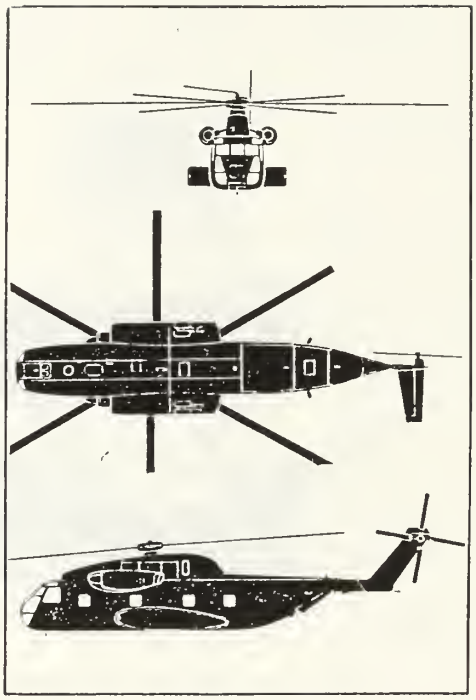
Wings: no
Engine: twin turboshaft
Horizontal Stabilizer:
end-mounted on tail boom and full
Fuselage: fail tail boom
Undercarriage: skid
Main Rotor: single
Tail Rotor Placement:
external

CH-47 Chinook



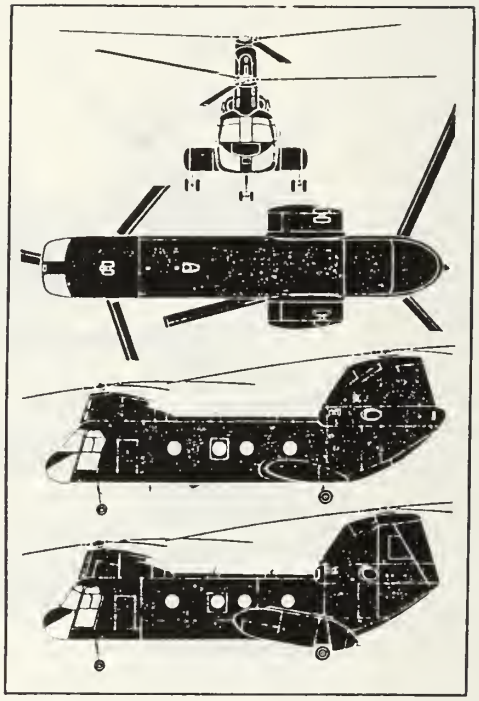
Wings: no
Engine: twin turboshaft
Horizontal Stabilizer:
no horizontal stabilizer
Fuselage: no tail boom
Undercarriage: wheeled
Main Rotor: twin
Tail Rotor Placement:
no tail rotor

CH-53 Sea Stallion



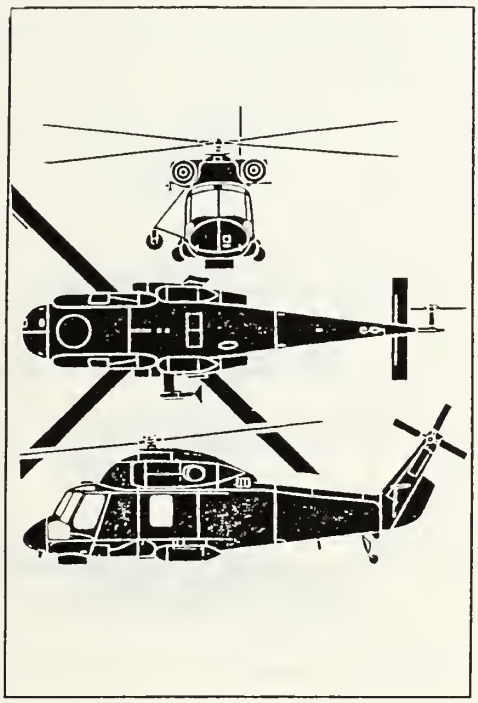
Wings: in stub form
Engine: twin turboshaft
Horizontal Stabilizer:
mounted on vertical
stabilizer and half
Fuselage: fair tail boom
Undercarriage: retractable
Tail Rotor Placement:
externaln

H-46 Sea Knight



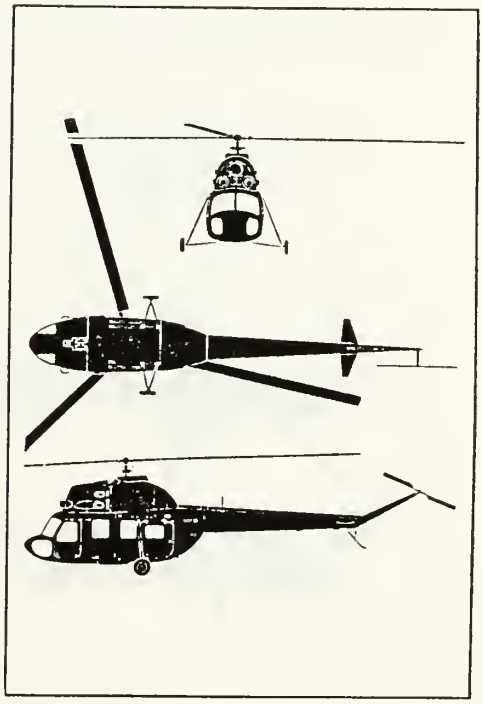
Wings: no
Engine: twin turboshaft
Horizontal Stabilizer:
no horizontal stabilizer
Fuselage: no tail boom
Undercarriage: wheeled
Main Rotor: twin
Tail Rotor Placement:
no tail rotor

H-2 Seasprite



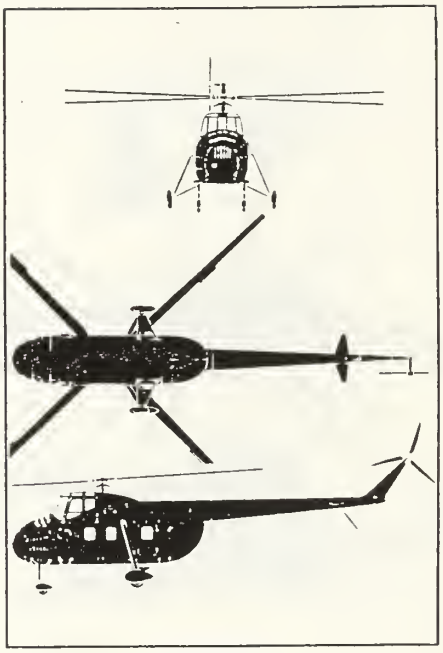
Wings: in stub form
Engine: twin turboshaft
Horizontal Stabilizer:
end-mounted on tail boom
and full
Fuselage: fair tail boom
Undercarriage: retractable
Main Rotor: single
Tail Rotor Placement:
external

Mi-2 Hoplite



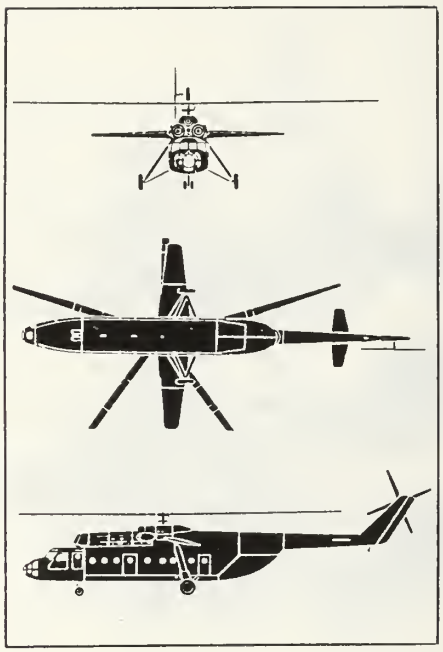
Wings: no
Engine: twin turboshaft
Horizontal Stabilizer:
end-mounted on tail boom
and full
Fuselage: fair tail boom
Undercarriage: wheeled
Main Rotor: single
Tail Rotor Placement:
external

Mi-4 Hound



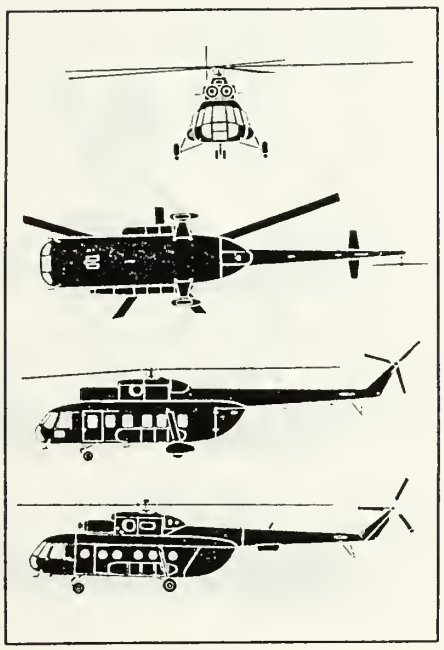
Wings: no
Engine: single piston
Horizontal Stabilizer:
end-mounted on tail boom
Fuselage: fair tail boom
Undercarriage: wheeled
Main Rotor: single
Tail Rotor Placement:
external

Mi-6 Hook



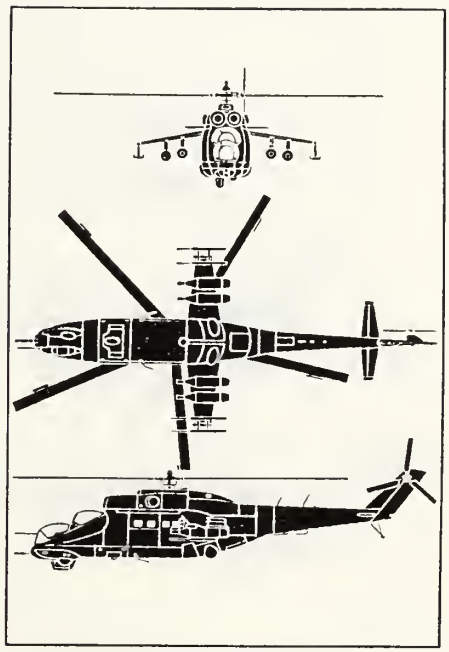
Wings: in stub form
Engine: twin turboshaft
Horizontal Stabilizer:
end-mounted on tail boom
Fuselage: fair tail boom
Undercarriage: wheeled
Main Rotor: single
Tail Rotor Placement:
external

Mi-8 Hip



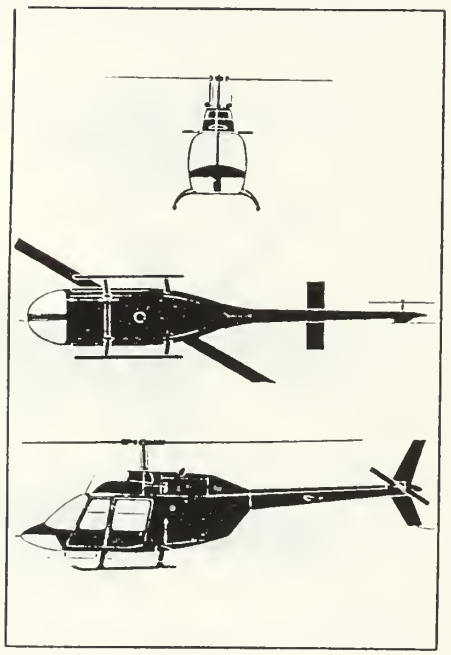
Wings: in stub form
Engine: twin turboshaft
Horizontal Stabilizer:
mid-mounted on tail boom and full
Fuselage: fair tail boom
Undercarriage: wheeled
Main Rotor: single
Tail Rotor Placement:
external

Mi-24 Hind



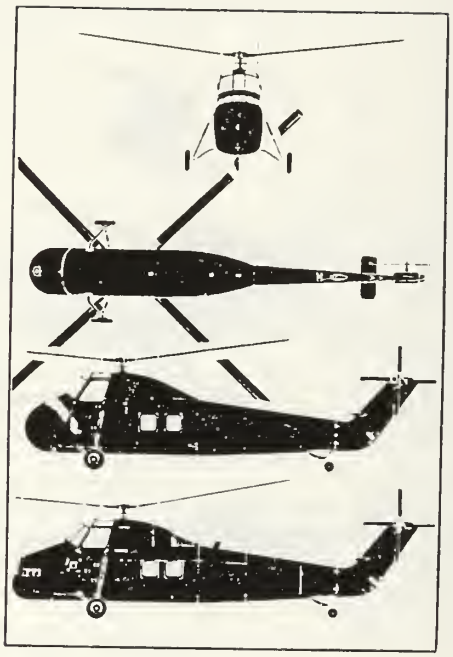
Wings: in stub form
Engine: twin turboshaft
Horizontal Stabilizer:
end-mounted on tail boom and full
Fuselage: fair tail boom
Undercarriage: retractable
Main Rotor: single
Tail Rotor Placement:
external

OH-58 Kiowa



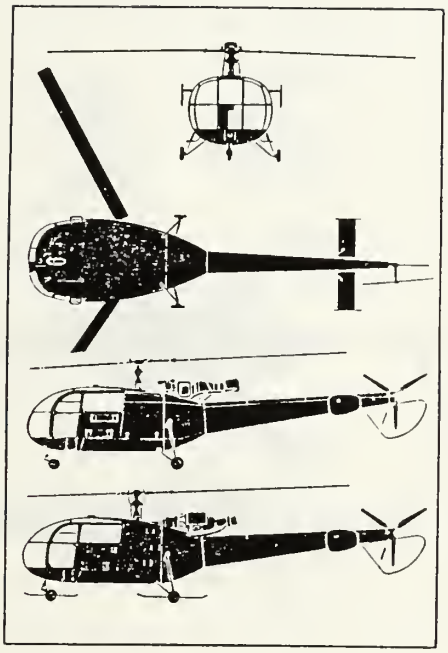
Wings: no
Engine: single turboshaft
Horizontal Stabilizer:
mid-mounted on tail boom
and full
Fuselage: fair tail boom
Undercarriage: skid
Main Rotor: single
Tail Rotor Placement:
external

S-58T



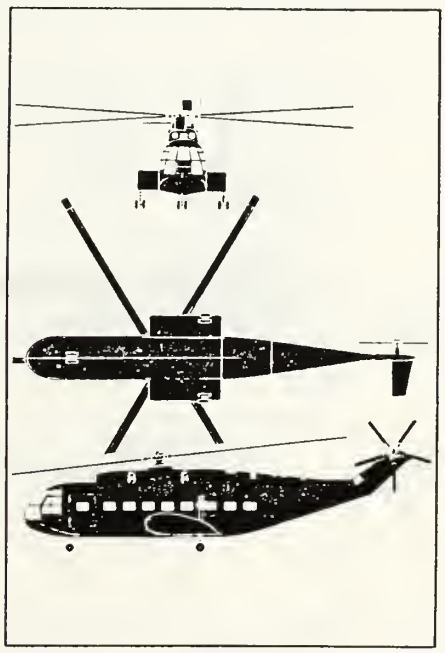
Wings: no
Engine: single turboshaft
Horizontal Stabilizer:
mounted on vertical
stabilizer and full
Fuselage: fair tail boom
Undercarriage: wheeled
Main Rotor: single
Tail Rotor Placement:
external

SA316



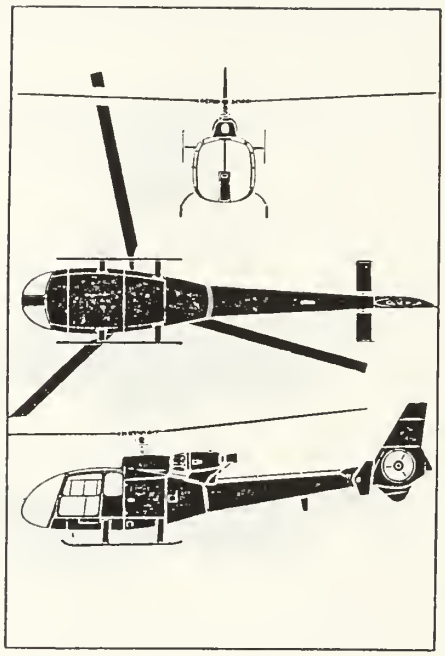
Wings: no
Engine: single turboshaft
Horizontal Stabilizer:
end-mounted on tail boom
and full
Fuselage: fair tail boom
Undercarriage: wheeled
Main Rotor: single
Tail Rotor Placement:
external

SA321



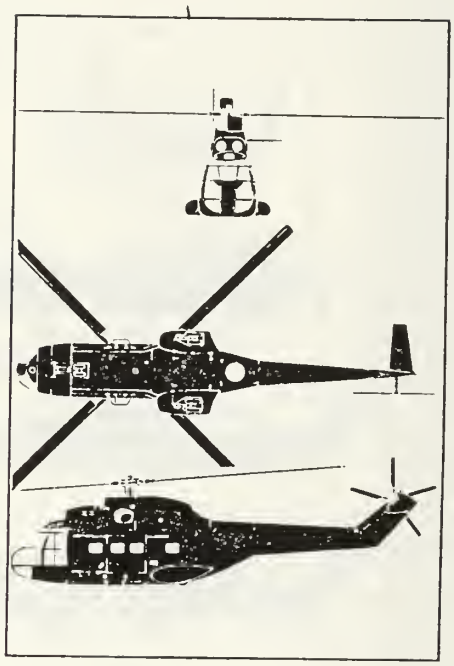
Wings: in stub form
Engine: three turboshaft
Horizontal Stabilizer:
mounted on vertical
stabilizer and half
Fuselage: fair tail boom
Undercarriage: wheeled
Main Rotor: single
Tail Rotor Placement:
external

SA341



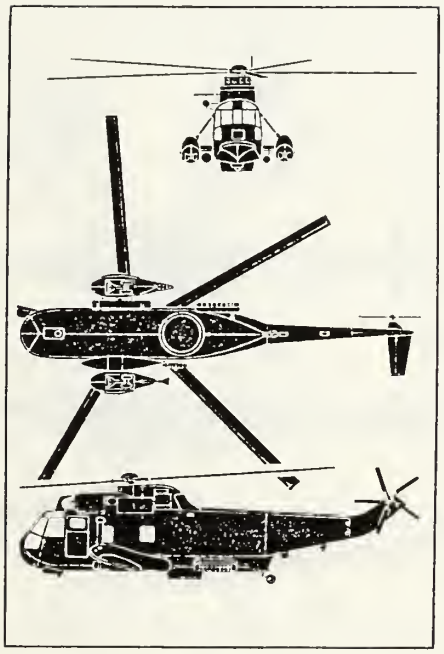
Wings: no
Engine: single turboshaft
Horizontal Stabilizer:
end-mounted on tail boom
Fuselage: fair tail boom
Undercarriage: skid
Main Rotor: single
Tail Rotor Placement:
inside tail boom

SA330 Puma



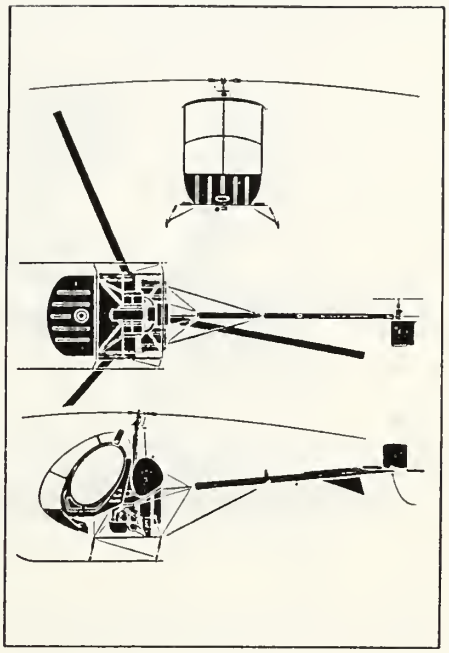
Wings: no
Engine: twin turboshaft
Horizontal Stabilizer:
end-mounted on tail boom
and half
Undercarriage: retractable
Main Rotor: single
Tail Rotor Placement:
external

Sikorsky SH-3H



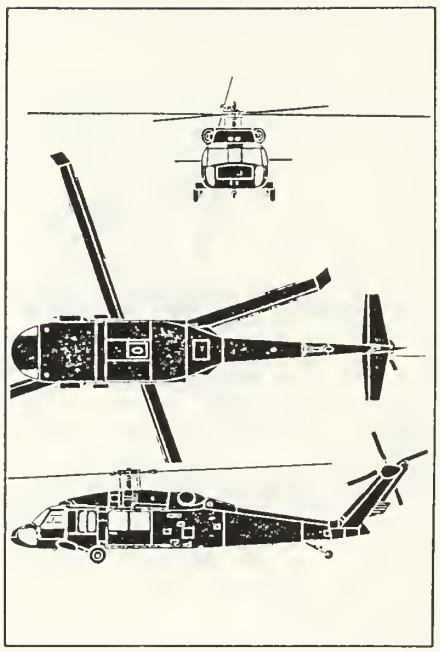
Wings: no
Engine: twin turboshaft
Horizontal Stabilizer:
mounted on vertical
stabilizer and half
Fuselage: fair tail boom
Undercarriage: retractable
Main Rotor: single
Tail Rotor Placement:
external

TH-55 Osage



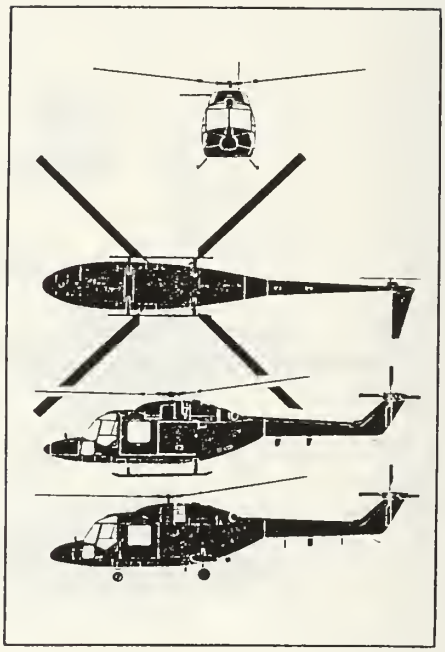
Wings: no
Engine: single piston
Horizontal Stabilizer:
end-mounted on tail boom
and half
Fuselage: fair tail boom
Undercarriage: skid
Main Rotor: single
Tail Rotor Placement:
external

UH-60 Black Hawk



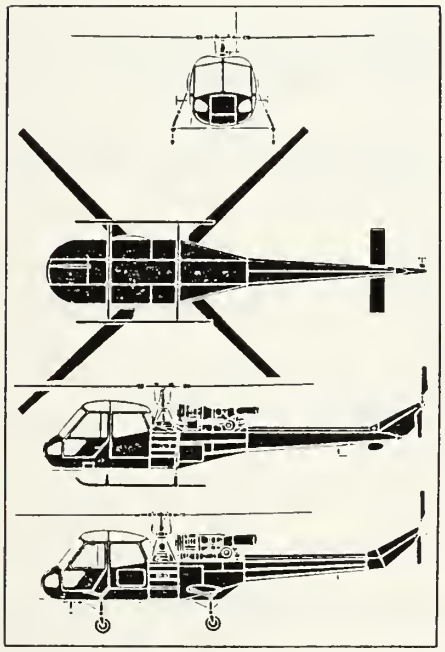
Wings: no
Engine: twin turboshaft
Horizontal Stabilizer:
end-mounted on tail boom and full
Fuselage: fair tail boom
Undercarriage: wheeled
Main Rotor: single
Tail Rotor Placement:
external

Westland Lynx



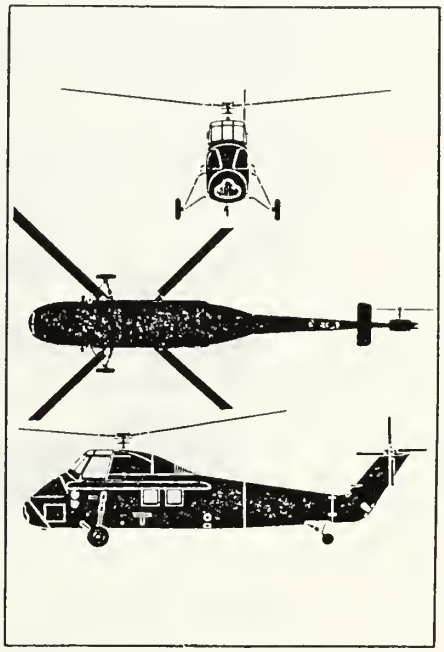
Wings: no
Engine: twin turboshaft
Horizontal Stabilizer:
mounted on vertical stabilizer and half
Fuselage: fair tail boom
Undercarriage: skid
Main Rotor: single
Tail Rotor Placement:
external

Westland Scout



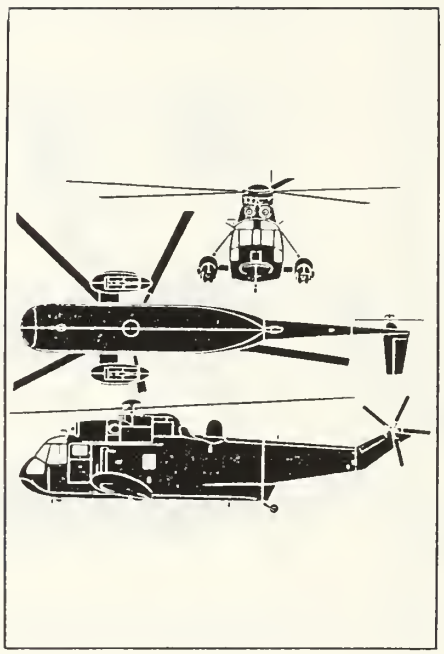
Wings: no
Engine: single turboshaft
Horizontal Stabilizer:
end-mounted on tail boom and full
Fuselage: fair tail boom
Undercarriage: skid
Main Rotor: single
Tail Rotor Placement:
external

HAS1



Wings: no
Engine: twin turboshaft
Horizontal Stabilizer:
end-mounted on tail boom and full
Fuselage: fair tail boom
Undercarriage: wheeled
Main Rotor: single
Tail Rotor Placement:
external

HAS2 Sea King



Wings: no
Engine: twin turboshaft
Horizontal Stabilizer:
mounted on vertical
stabilizer and half
Fuselage: fair tail boom
Undercarriage: retractable
Main Rotor: single
Tail Rotor Placement:
external

## LIST OF REFERENCES


1. Sleeman, D., "Intelligent Tutoring Systems: A Review," School Education and Department of Computer Science, Stanford University, California, 1984.

2. Pliske, D. B. and Psotka, J., "Exploratory Programming Environments for Designing ICAI," *Journal of Computer-Based Instruction*, v. 13, Spring 1986.

3. Jones, M., "Application of Artificial Intelligent within Education," p. 517, *Comp. and Math. with Appls.*, v. 11, 1985.

4. Woolf, B. P., "Representing Complex Knowledge in an Intelligent Machine Tutor." *Computational Intelligence*, v. 3, 1987.

5. Wenger, E., *Artificial Intelligence and Tutoring Systems*, Morgan Kaufmann Publishers, 1987.

6. Kearsley, G., *Artificial Intelligent and Instruction*, p. 17, Addison-Wesley Publishing Company, 1987.

7. Barr, A. and Feigenbaum, E. A., *The Handbook of Artificial Intelligence*, pp 225-294, v. 2, William Kaufmann, 1982.

8. Woolf, B., *Intelligent Tutoring Systems: A Survey,* pp 1-44, Morgan Kaufmann Publishers, 1988.

9. Chang, T. K., *Helicopter*, pp 20-35, World Publishing Company, 1977.

10. Campbell, L. W., *An Intelligent Tutor System for Visual Aircraft Recognition*, Master's Thesis, Naval Postgraduate School, Monterey, California, June, 1990.

11. Wood, D., *Jane's World Aircraft Recognition Handbook*, Jane's Information Group, 1985.

12. Borland International, Inc., *Turbo Pascal*, v. 6.0, 1991.

13. Park, O., "Functional Characteristics of Intelligent Computer-Assisted Instruction: Intelligent Features," *Educational Technology*, v.28, June 1988.

## INITIAL DISTRIBUTION LIST

| | | |
|---|---|---|
| 1. | Defense Technical Information Center<br>Cameron Station<br>Alexandria, VA 22304-6145 | 2 |
| 2. | Library, Code 052<br>Naval Postgraduate School<br>Monterey, CA 93943-5002 | 2 |
| 3. | Department Chairman, Code CS<br>Department of Computer Science<br>Naval Postgraduate School<br>Monterey, CA 93943-5000 | 2 |
| 4. | Dr. Yuh-jeng Lee, Code CS/Le<br>Department of Computer Science<br>Naval Postgraduate School<br>Monterey, CA 93943-5100 | 3 |
| 5. | Dr. Man-Tak Shing, Code CS/Sh<br>Department of Computer Science<br>Naval Postgraduate School<br>Monterey, CA 93943-5100 | 1 |
| 6. | Department Chairman<br>Department of General Courses<br>Chung-Cheng Institute of Technology<br>Tao-Yuan, Taiwan, Republic of China | 1 |
| 7. | Capt. Ling, Ming-Tien<br>Department of General Courses<br>Chung-Cheng Institute of Technology<br>Tao-Yuan, Taiwan, Republic of China | 3 |
| 8. | Library<br>Chung-Cheng Institute of Technology<br>Tao-Yuan, Taiwan, Republic of China | 1 |

Thesis
L66335 Ling
c.1 An intelligent training system for helicopter recognition.